- Jeremy Beard
- Due 20220410
- Week 5
- MSDS650
- Genie Hays

Week 5 Lab: Supervised Learning

This week's assignment will focus completeing a KNN analysis and comparing its performance with other supervised algorithms.

Our Dataset:

Dataset: bank-additional-full.csv (Provided in folder assign_wk5)

Remember to take a look at the bank-additional-names.txt files for a better understanding of the dataset.

Assignment Requirements

Part 1: KNN Analysis

Objective: According to the dataset's text file, the target column the last column in the dataset.

- Cleanup the dataset as you deem appropriate. As always, defend your reasoning!!!
    - Missing values?
    - Column names
- Prepare the data for machine learning
    - A little EDA goeas a long way
    - Do you need to do anything about data types?
- KNN analysis
    - What is your objective from the analysis?
    - What is your optimal K?
    - How about accuracy rate?
        - Discover any insights from this analysis?

- Include numbers/graphs corresponding to your conclusions
    - Discuss ways to improve the performance of your KNN model
    - Defend and backup your thoughts!!!!!!

Part 2: Comparison to other supervised algorithm

As we saw in the lecture notebook, algorithm performance varies based on the algorithm used. The lecture demostrated using K-Fold Cross-Validation to compare the performance of several algorithm for the same dataset.

- At the end of part 1 you discussed ways to improve the performance of you KNN model.
    - Implement one of those methods to improve your KNN model performance.
    - Rerun a KNN analysis for your improved dataset
    - Discuss the change in performance from the model in part 1
- Complete a K-fold cross-validation analysis for your improved model
    - You need to use at less three additional models
    - Discuss the difference in the performance of the 4 algorithms against your improved dataset.

Deliverables:

Upload your Jupyter Notebook to the corresponding location in WorldClass.

Note:: Make sure you have clearly indicated each assignment requirement within your notebook.

Important: Make sure your provide complete and thorough explanations for all of your analysis. You need to defend your thought processes and reasoning.

I. Introduction

Throughout the course of this assignment

II. Methods, III. Code, and IV. Analysis of Results

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
%matplotlib inline
sns.set()
```

```
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
#Cleanup the dataset as you deem appropriate. As always, defend your reasoning!!!
##Missing values?
##Column names
df = pd.read_csv('assign_wk5/bank-additional-full.csv', sep=';')
df.head(20)
```

Out[2]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	0	nonexistent
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	0	nonexistent
5	45	services	married	basic.9y	unknown	no	no	telephone	may	mon	...	1	999	0	nonexistent
6	59	admin.	married	professional.course	no	no	no	telephone	may	mon	...	1	999	0	nonexistent
7	41	blue-collar	married	unknown	unknown	no	no	telephone	may	mon	...	1	999	0	nonexistent
8	24	technician	single	professional.course	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent
9	25	services	single	high.school	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent
10	41	blue-collar	married	unknown	unknown	no	no	telephone	may	mon	...	1	999	0	nonexistent
11	25	services	single	high.school	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent
12	29	blue-collar	single	high.school	no	no	yes	telephone	may	mon	...	1	999	0	nonexistent
13	57	housemaid	divorced	basic.4y	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent
14	35	blue-collar	married	basic.6y	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent
15	54	retired	married	basic.9y	unknown	yes	yes	telephone	may	mon	...	1	999	0	nonexistent
16	35	blue-collar	married	basic.6y	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent
17	46	blue-collar	married	basic.6y	unknown	yes	yes	telephone	may	mon	...	1	999	0	nonexistent
18	50	blue-collar	married	basic.9y	no	yes	yes	telephone	may	mon	...	1	999	0	nonexistent
19	39	management	single	basic.9y	unknown	no	no	telephone	may	mon	...	1	999	0	nonexistent

20 rows × 21 columns

First, let's just get some information about the data! With info(), and shape. Describe() will also be used later on.

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             41188 non-null  int64  
 1   job             41188 non-null  object 
 2   marital         41188 non-null  object 
 3   education       41188 non-null  object 
 4   default         41188 non-null  object 
 5   housing         41188 non-null  object 
 6   loan            41188 non-null  object 
 7   contact         41188 non-null  object 
 8   month           41188 non-null  object 
 9   day_of_week     41188 non-null  object 
 10  duration        41188 non-null  int64  
 11  campaign        41188 non-null  int64  
 12  pdays           41188 non-null  int64  
 13  previous        41188 non-null  int64  
 14  poutcome        41188 non-null  object 
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object 
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

In [4]:

```
df.shape
```

Out[4]:

```
(41188, 21)
```

Next, let's rename the columns to something a bit more descriptive. After looking at the text file included, the following column titles seemed appropriate:

In [5]:

```
#I want to clarify some column names:
# default --> has_credit_in_default
# housing --> has_housing_loan
```

```python
# loan --> has_personal_loan
# contact --> contact_type
# month --> last_contact_month
# day_of_week --> last_contact_day_of_week
# duration --> last_contact_duration
# campaign --> num_contacts_this_campaign
# pdays --> num_days_prev_campaign
# previous --> num_contacts_prev_campaign
# poutcome --> outcome_prev_campaign
# y --> subbed_term_deposit
df.rename(columns = {'default':'has_credit_in_default', \
                     'housing':'has_housing_loan', \
                     'loan':'has_personal_loan', \
                     'contact':'contact_type', \
                     'month':'last_contact_month', \
                     'day_of_week':'last_contact_day_of_week', \
                     'duration':'last_contact_duration', \
                     'campaign':'num_contacts_this_campaign', \
                     'pdays':'num_days_prev_campaign', \
                     'previous':'num_contacts_prev_campaign', \
                     'poutcome':'outcome_prev_campaign', \
                     'y':'subbed_term_deposit'}, \
          inplace = True)

df.head(20)
```

Out[5]:

	age	job	marital	education	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact_
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	
1	57	services	married	high.school	unknown	no	no	telephone	may	
2	37	services	married	high.school	no	yes	no	telephone	may	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	
4	56	services	married	high.school	no	no	yes	telephone	may	
5	45	services	married	basic.9y	unknown	no	no	telephone	may	
6	59	admin.	married	professional.course	no	no	no	telephone	may	
7	41	blue-collar	married	unknown	unknown	no	no	telephone	may	
8	24	technician	single	professional.course	no	yes	no	telephone	may	
9	25	services	single	high.school	no	yes	no	telephone	may	
10	41	blue-collar	married	unknown	unknown	no	no	telephone	may	
11	25	services	single	high.school	no	yes	no	telephone	may	

	age	job	marital	education	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact_
12	29	blue-collar	single	high.school	no	no	yes	telephone	may	
13	57	housemaid	divorced	basic.4y	no	yes	no	telephone	may	
14	35	blue-collar	married	basic.6y	no	yes	no	telephone	may	
15	54	retired	married	basic.9y	unknown	yes	yes	telephone	may	
16	35	blue-collar	married	basic.6y	no	yes	no	telephone	may	
17	46	blue-collar	married	basic.6y	unknown	yes	yes	telephone	may	
18	50	blue-collar	married	basic.9y	no	yes	yes	telephone	may	
19	39	management	single	basic.9y	unknown	no	no	telephone	may	

20 rows × 21 columns

In [6]:

```
df.columns
```

Out[6]:

```
Index(['age', 'job', 'marital', 'education', 'has_credit_in_default',
       'has_housing_loan', 'has_personal_loan', 'contact_type',
       'last_contact_month', 'last_contact_day_of_week',
       'last_contact_duration', 'num_contacts_this_campaign',
       'num_days_prev_campaign', 'num_contacts_prev_campaign',
       'outcome_prev_campaign', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed', 'subbed_term_deposit'],
      dtype='object')
```

In [7]:

```
df.isnull().sum()
```

Out[7]:

```
age                           0
job                           0
marital                       0
education                     0
has_credit_in_default         0
has_housing_loan              0
has_personal_loan             0
contact_type                  0
last_contact_month            0
last_contact_day_of_week      0
last_contact_duration         0
num_contacts_this_campaign    0
num_days_prev_campaign        0
num_contacts_prev_campaign    0
outcome_prev_campaign         0
```

```
emp.var.rate                0
cons.price.idx              0
cons.conf.idx               0
euribor3m                   0
nr.employed                 0
subbed_term_deposit         0
dtype: int64
```

So, there are no null values! This is good right? Well, it's not exactly true, since for this dataset, 'unknown' is used to indicate a missing value.

Now let's make plots of all the variables to get a sense of what they look like, and what must be done for data cleaning!

In [8]:

```python
df['age'].value_counts()
```

Out[8]:

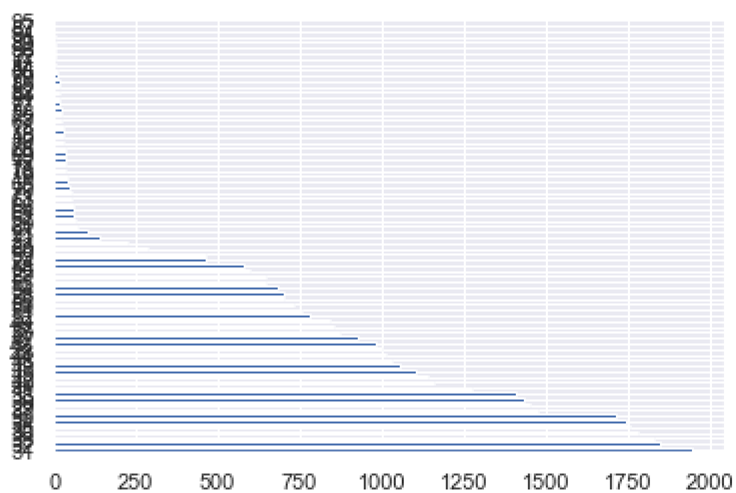
```
31    1947
32    1846
33    1833
36    1780
35    1759
      ...
89       2
91       2
94       1
87       1
95       1
Name: age, Length: 78, dtype: int64
```

In [9]:

```python
df['age'].value_counts().plot.barh()
```

Out[9]:

```
<AxesSubplot:>
```



In [ ]:

In [10]:

```
df['job'].value_counts()
```

Out[10]:

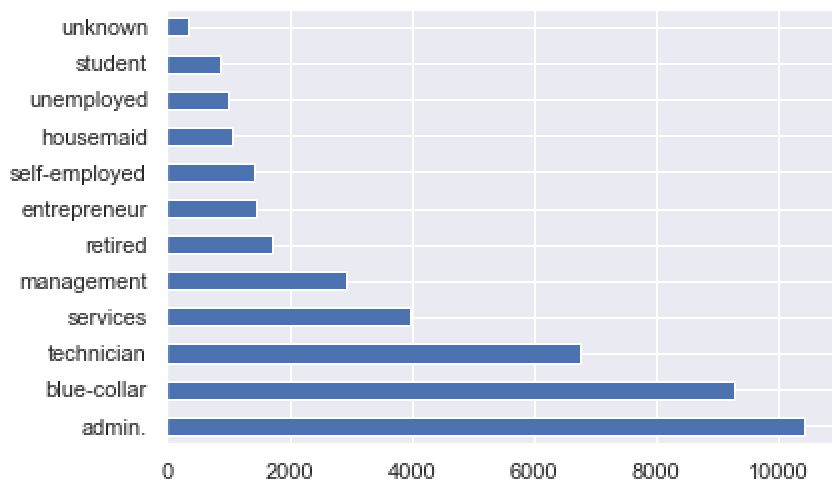
```
admin.           10422
blue-collar       9254
technician        6743
services          3969
management        2924
retired           1720
entrepreneur      1456
self-employed     1421
housemaid         1060
unemployed        1014
student            875
unknown            330
Name: job, dtype: int64
```

In [11]:

```
df['job'].value_counts().plot.barh()
```

Out[11]:

```
<AxesSubplot:>
```



In [12]:

```
df['job'].value_counts()['unknown']
```

Out[12]:

```
330
```

In [13]:

```
df['marital'].value_counts()
```

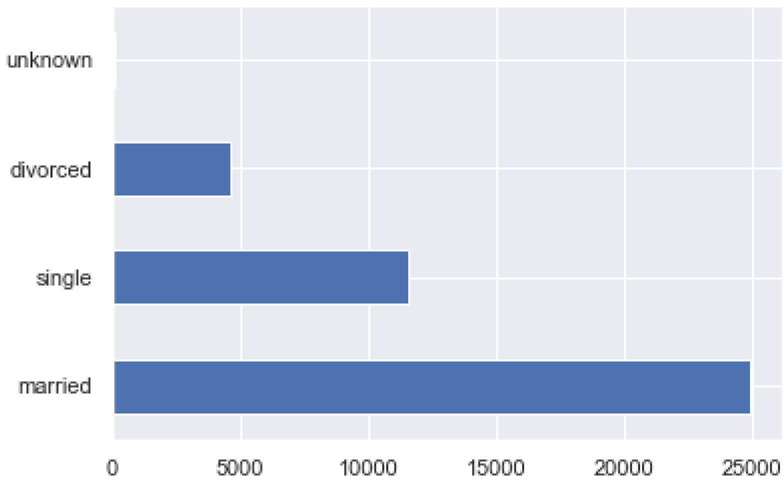
Out[13]:

```
married     24928
single      11568
```

```
divorced     4612
unknown        80
Name: marital, dtype: int64
```

In [14]:
```python
df['marital'].value_counts().plot.barh()
```

Out[14]:
```
<AxesSubplot:>
```



In [15]:
```python
df['marital'].value_counts()['unknown']
```

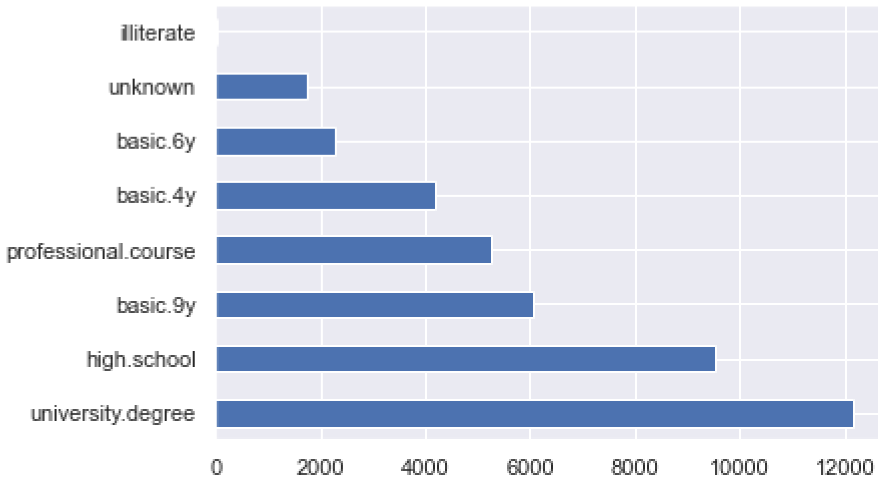
Out[15]:
```
80
```

In [16]:
```python
df['education'].value_counts()
```

Out[16]:
```
university.degree      12168
high.school             9515
basic.9y                6045
professional.course     5243
basic.4y                4176
basic.6y                2292
unknown                 1731
illiterate                18
Name: education, dtype: int64
```

In [17]:
```python
df['education'].value_counts().plot.barh()
```

Out[17]:
```
<AxesSubplot:>
```



In [18]:
```
df['education'].value_counts()['unknown']
```

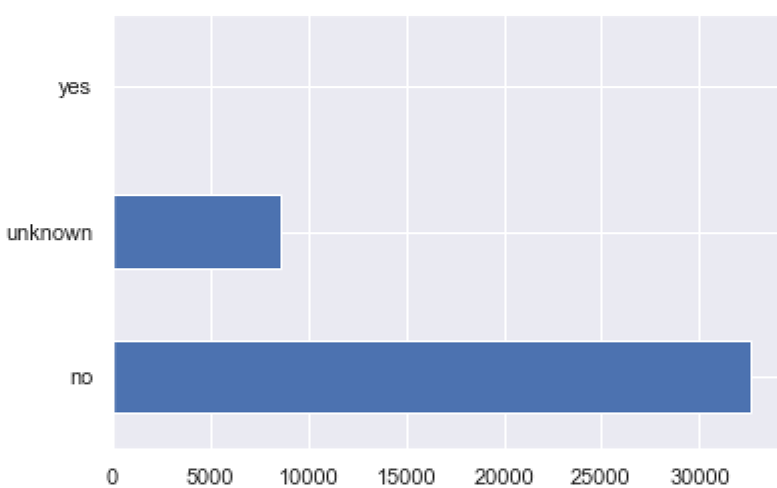
Out[18]:
```
1731
```

In [19]:
```
df['has_credit_in_default'].value_counts()
```

Out[19]:
```
no         32588
unknown     8597
yes            3
Name: has_credit_in_default, dtype: int64
```

In [20]:
```
df['has_credit_in_default'].value_counts().plot.barh()
```

Out[20]:
```
<AxesSubplot:>
```



In [21]:

```
df['has_credit_in_default'].value_counts()['unknown']
```

Out[21]:

```
8597
```

In [22]:

```
df['has_housing_loan'].value_counts()
```

Out[22]:

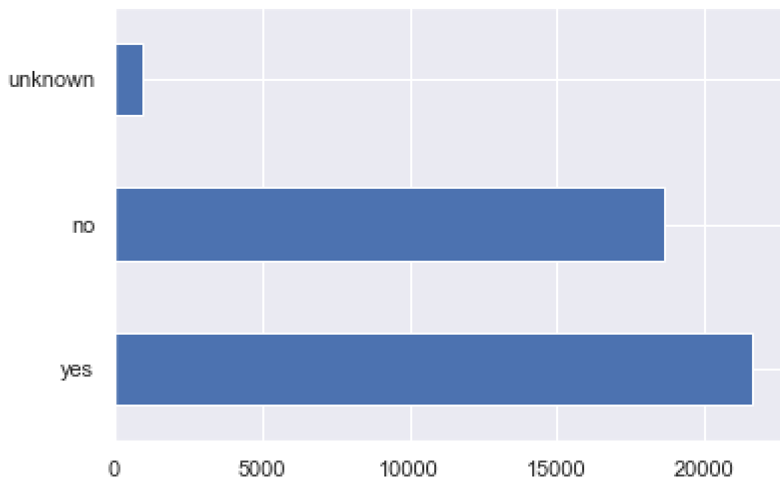
```
yes        21576
no         18622
unknown      990
Name: has_housing_loan, dtype: int64
```

In [23]:

```
df['has_housing_loan'].value_counts().plot.barh()
```

Out[23]:

```
<AxesSubplot:>
```



In [24]:

```
df['has_housing_loan'].value_counts()['unknown']
```

Out[24]:

```
990
```

In [25]:

```
df['has_personal_loan'].value_counts()
```

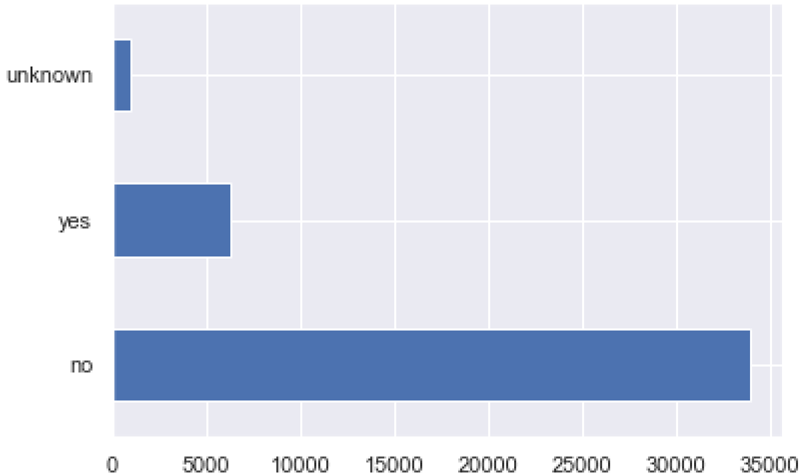
Out[25]:

```
no         33950
yes         6248
unknown      990
Name: has_personal_loan, dtype: int64
```

In [26]:

```
df['has_personal_loan'].value_counts().plot.barh()
```

Out[26]: <AxesSubplot:>



In [27]:
```python
df['has_personal_loan'].value_counts()['unknown']
```

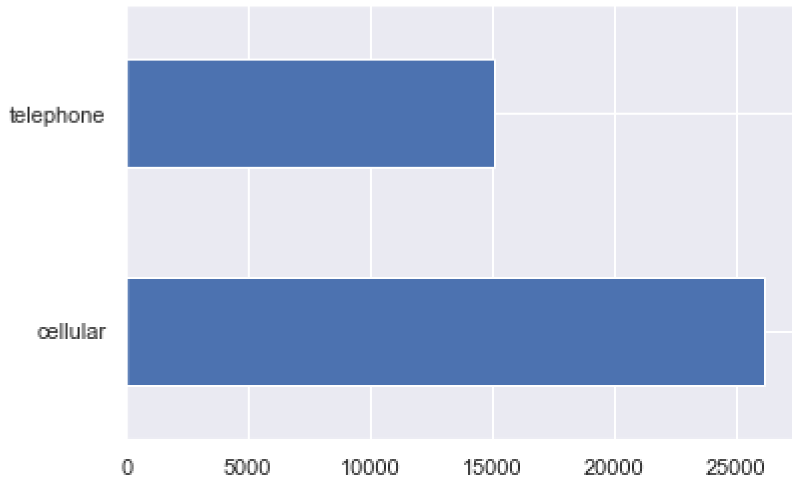
Out[27]:
```
990
```

In [28]:
```python
df['contact_type'].value_counts()
```

Out[28]:
```
cellular     26144
telephone    15044
Name: contact_type, dtype: int64
```

In [29]:
```python
df['contact_type'].value_counts().plot.barh()
```

Out[29]: <AxesSubplot:>



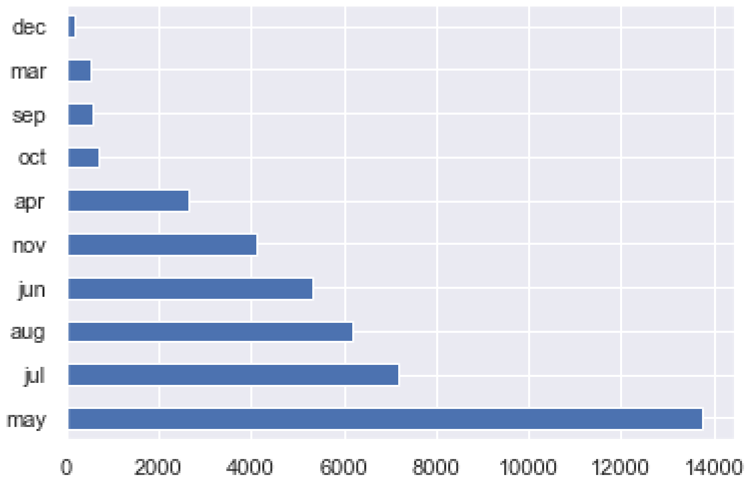
In [ ]:

In [30]:
```
df['last_contact_month'].value_counts()
```

Out[30]:
```
may    13769
jul     7174
aug     6178
jun     5318
nov     4101
apr     2632
oct      718
sep      570
mar      546
dec      182
Name: last_contact_month, dtype: int64
```

In [31]:
```
df['last_contact_month'].value_counts().plot.barh()
```

Out[31]:
```
<AxesSubplot:>
```

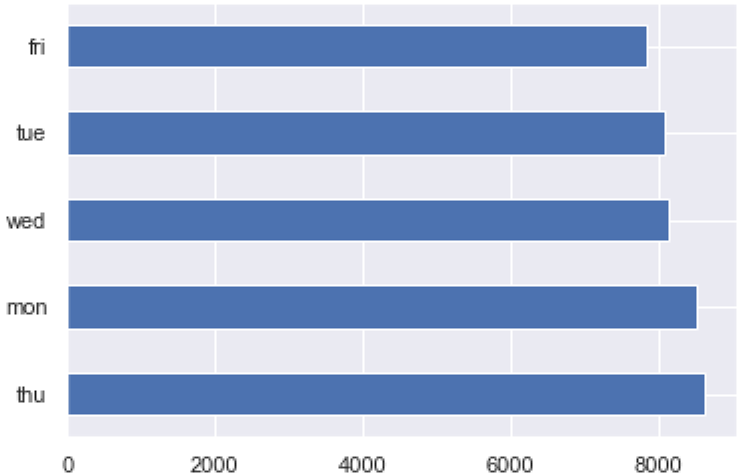


In [32]:
```
df['last_contact_day_of_week'].value_counts()
```

Out[32]:
```
thu    8623
mon    8514
wed    8134
tue    8090
fri    7827
Name: last_contact_day_of_week, dtype: int64
```

In [33]:
```
df['last_contact_day_of_week'].value_counts().plot.barh()
```

Out[33]: <AxesSubplot:>

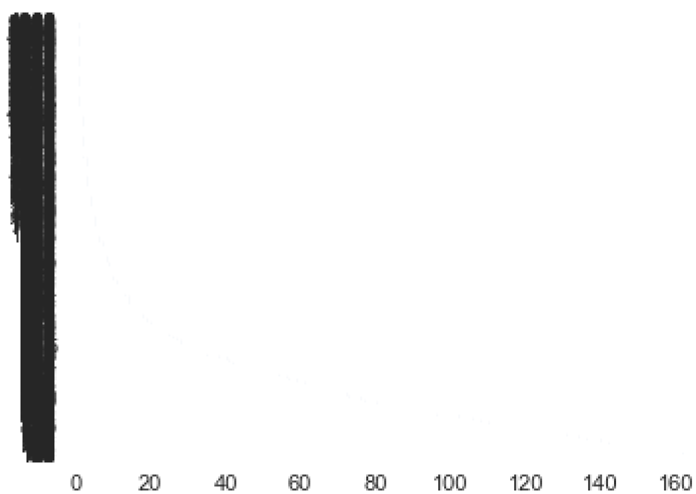


In [34]:
```
df['last_contact_duration'].value_counts()
```

Out[34]:
```
90      170
85      170
136     168
73      167
124     164
       ...
1569      1
1053      1
1263      1
1169      1
1868      1
Name: last_contact_duration, Length: 1544, dtype: int64
```

In [35]:
```python
df['last_contact_duration'].value_counts().plot.barh()
```

Out[35]:
```
<AxesSubplot:>
```



In [36]:
```python
df['num_contacts_this_campaign'].value_counts()
```

Out[36]:
```
1     17642
2     10570
3      5341
4      2651
5      1599
6       979
7       629
8       400
9       283
10      225
11      177
12      125
13       92
```

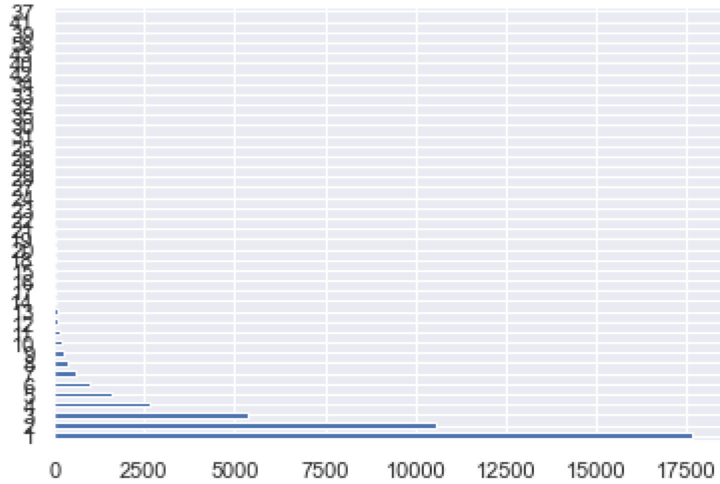
```
14      69
17      58
16      51
15      51
18      33
20      30
19      26
21      24
22      17
23      16
24      15
27      11
29      10
28       8
26       8
25       8
31       7
30       7
35       5
32       4
33       4
34       3
42       2
40       2
43       2
56       1
39       1
41       1
37       1
Name: num_contacts_this_campaign, dtype: int64
```

In [37]:

```
df['num_contacts_this_campaign'].value_counts().plot.barh()
```

Out[37]:

```
<AxesSubplot:>
```



In [38]:

```
df['num_days_prev_campaign'].value_counts()
```

Out[38]:

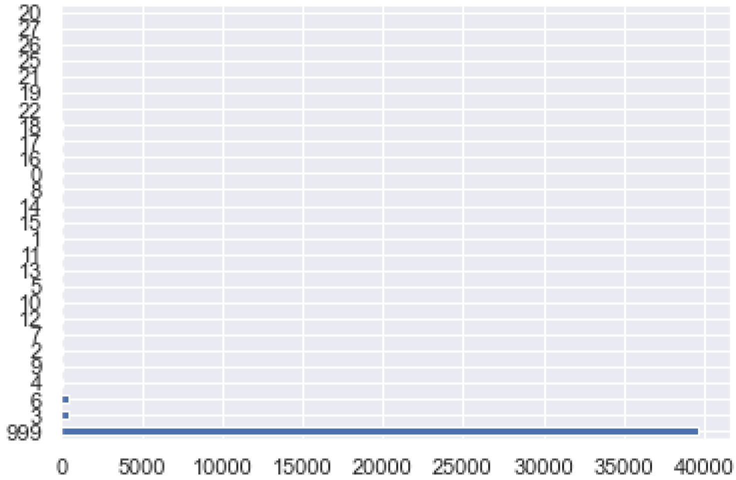
```
999    39673
3        439
6        412
4        118
9         64
2         61
7         60
12        58
10        52
5         46
13        36
11        28
1         26
15        24
14        20
8         18
0         15
16        11
17         8
18         7
22         3
19         3
21         2
25         1
26         1
27         1
20         1
Name: num_days_prev_campaign, dtype: int64
```

In [39]:

```
df['num_days_prev_campaign'].value_counts().plot.barh()
```

Out[39]:

```
<AxesSubplot:>
```



In [40]:

```python
df['num_contacts_prev_campaign'].value_counts()
```

Out[40]:

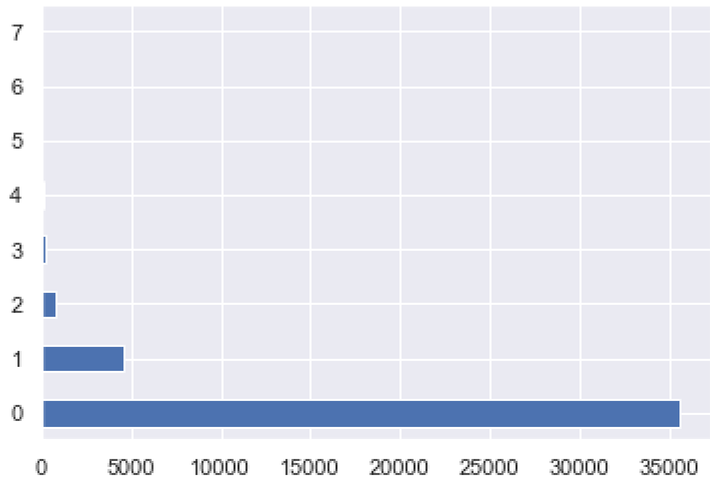
```
0    35563
1     4561
2      754
3      216
4       70
5       18
6        5
7        1
Name: num_contacts_prev_campaign, dtype: int64
```

In [41]:

```python
df['num_contacts_prev_campaign'].value_counts().plot.barh()
```

Out[41]:

```
<AxesSubplot:>
```



In [42]:

```
df['outcome_prev_campaign'].value_counts()
```

Out[42]:

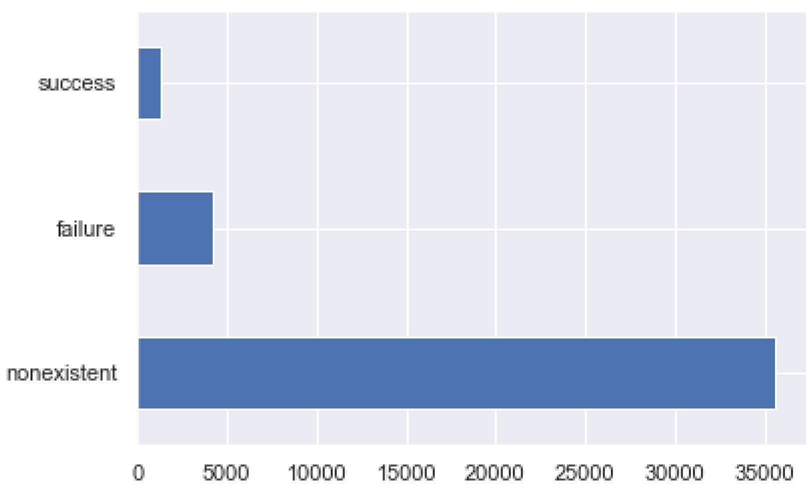
```
nonexistent    35563
failure         4252
success         1373
Name: outcome_prev_campaign, dtype: int64
```

In [43]:

```
df['outcome_prev_campaign'].value_counts().plot.barh()
```

Out[43]:

```
<AxesSubplot:>
```



In [44]:

```
df['emp.var.rate'].value_counts()
```

Out[44]:

```
 1.4    16234
-1.8     9184
```

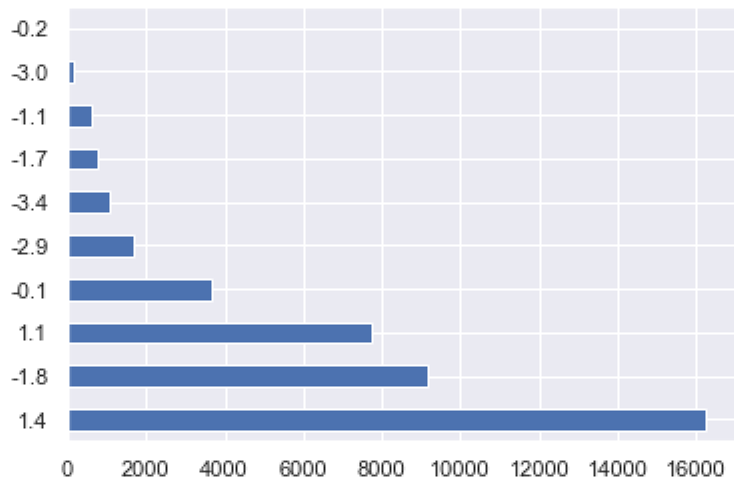
```
 1.1     7763
-0.1     3683
-2.9     1663
-3.4     1071
-1.7      773
-1.1      635
-3.0      172
-0.2       10
Name: emp.var.rate, dtype: int64
```

In [45]:

```python
df['emp.var.rate'].value_counts().plot.barh()
```

Out[45]:

```
<AxesSubplot:>
```



In [46]:

```python
df['cons.price.idx'].value_counts()
```

Out[46]:

```
93.994    7763
93.918    6685
92.893    5794
93.444    5175
94.465    4374
93.200    3616
93.075    2458
92.201     770
92.963     715
92.431     447
92.649     357
94.215     311
94.199     303
92.843     282
92.379     267
93.369     264
```

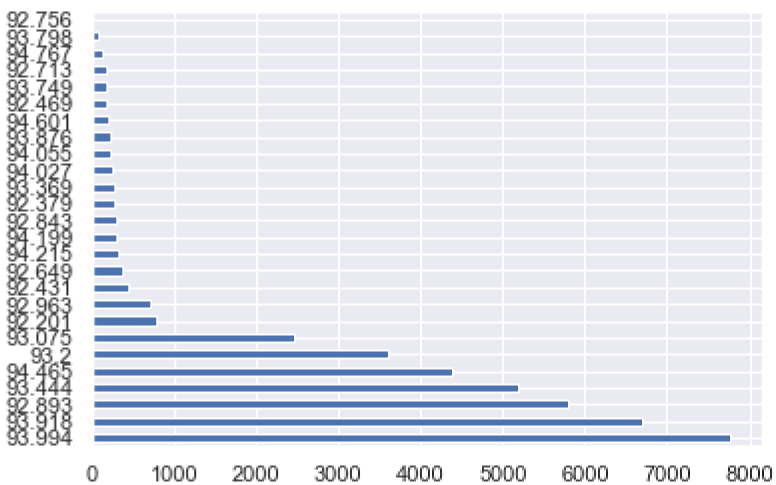
```
94.027     233
94.055     229
93.876     212
94.601     204
92.469     178
93.749     174
92.713     172
94.767     128
93.798      67
92.756      10
Name: cons.price.idx, dtype: int64
```

In [47]:

```python
df['cons.price.idx'].value_counts().plot.barh()
```

Out[47]:

```
<AxesSubplot:>
```



In [48]:

```python
df['cons.conf.idx'].value_counts()
```

Out[48]:

```
-36.4    7763
-42.7    6685
-46.2    5794
-36.1    5175
-41.8    4374
-42.0    3616
-47.1    2458
-31.4     770
-40.8     715
-26.9     447
-30.1     357
-40.3     311
-37.5     303
-50.0     282
```

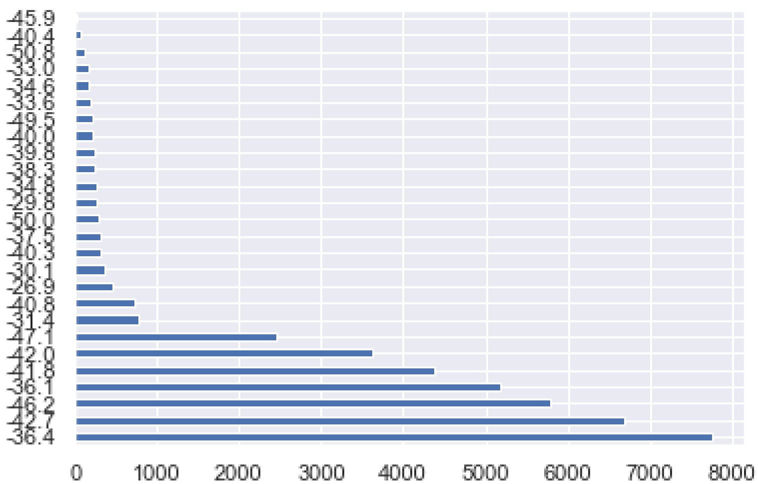
```
-29.8     267
-34.8     264
-38.3     233
-39.8     229
-40.0     212
-49.5     204
-33.6     178
-34.6     174
-33.0     172
-50.8     128
-40.4      67
-45.9      10
Name: cons.conf.idx, dtype: int64
```

In [49]:

```python
df['cons.conf.idx'].value_counts().plot.barh()
```

Out[49]:

```
<AxesSubplot:>
```



In [50]:

```python
df['euribor3m'].value_counts()
```

Out[50]:

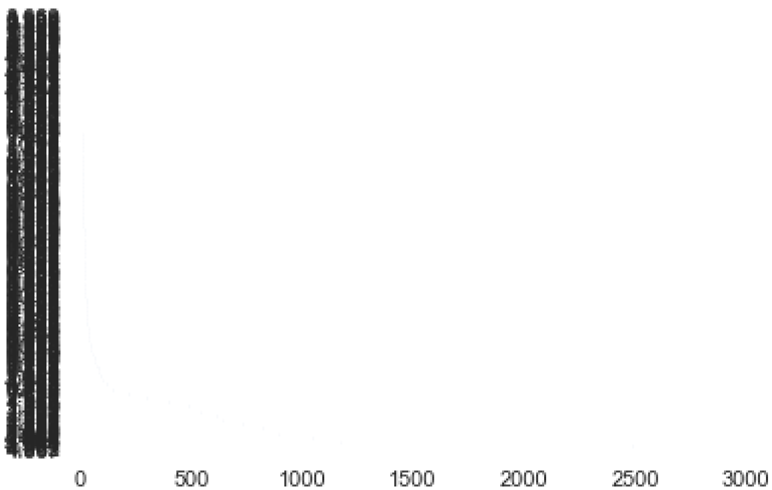
```
4.857    2868
4.962    2613
4.963    2487
4.961    1902
4.856    1210
         ...
3.853       1
3.901       1
0.969       1
0.956       1
3.669       1
Name: euribor3m, Length: 316, dtype: int64
```

In [51]:

```
df['euribor3m'].value_counts().plot.barh()
```

Out[51]:

```
<AxesSubplot:>
```



In [52]:

```
df['nr.employed'].value_counts()
```

Out[52]:

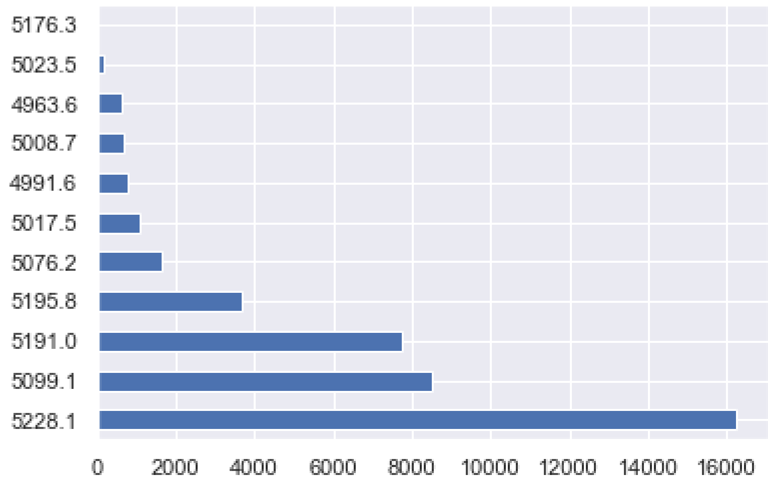
```
5228.1    16234
5099.1     8534
5191.0     7763
5195.8     3683
5076.2     1663
5017.5     1071
4991.6      773
5008.7      650
4963.6      635
5023.5      172
5176.3       10
Name: nr.employed, dtype: int64
```

In [53]:

```
df['nr.employed'].value_counts().plot.barh()
```

Out[53]:

```
<AxesSubplot:>
```

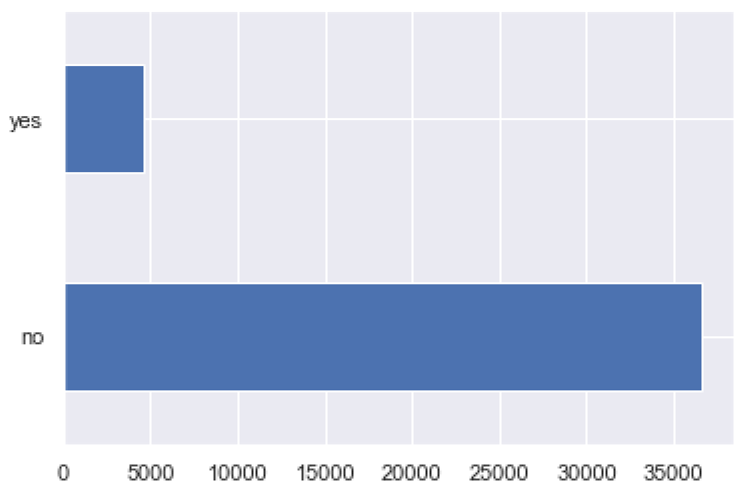


In [54]:
```
df['subbed_term_deposit'].value_counts()
```

Out[54]:
```
no     36548
yes     4640
Name: subbed_term_deposit, dtype: int64
```

In [55]:
```
df['subbed_term_deposit'].value_counts().plot.barh()
```

Out[55]:
```
<AxesSubplot:>
```



In [56]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
```

```
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   age                         41188 non-null  int64  
 1   job                         41188 non-null  object 
 2   marital                     41188 non-null  object 
 3   education                   41188 non-null  object 
 4   has_credit_in_default       41188 non-null  object 
 5   has_housing_loan            41188 non-null  object 
 6   has_personal_loan           41188 non-null  object 
 7   contact_type                41188 non-null  object 
 8   last_contact_month          41188 non-null  object 
 9   last_contact_day_of_week    41188 non-null  object 
 10  last_contact_duration       41188 non-null  int64  
 11  num_contacts_this_campaign  41188 non-null  int64  
 12  num_days_prev_campaign      41188 non-null  int64  
 13  num_contacts_prev_campaign  41188 non-null  int64  
 14  outcome_prev_campaign       41188 non-null  object 
 15  emp.var.rate                41188 non-null  float64
 16  cons.price.idx              41188 non-null  float64
 17  cons.conf.idx               41188 non-null  float64
 18  euribor3m                   41188 non-null  float64
 19  nr.employed                 41188 non-null  float64
 20  subbed_term_deposit         41188 non-null  object 
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

In [57]:

```python
df = df.replace('unknown', np.nan)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   age                         41188 non-null  int64  
 1   job                         40858 non-null  object 
 2   marital                     41108 non-null  object 
 3   education                   39457 non-null  object 
 4   has_credit_in_default       32591 non-null  object 
 5   has_housing_loan            40198 non-null  object 
 6   has_personal_loan           40198 non-null  object 
 7   contact_type                41188 non-null  object 
 8   last_contact_month          41188 non-null  object 
 9   last_contact_day_of_week    41188 non-null  object 
 10  last_contact_duration       41188 non-null  int64  
 11  num_contacts_this_campaign  41188 non-null  int64  
 12  num_days_prev_campaign      41188 non-null  int64  
 13  num_contacts_prev_campaign  41188 non-null  int64  
 14  outcome_prev_campaign       41188 non-null  object 
```

```
 15  emp.var.rate               41188 non-null  float64
 16  cons.price.idx             41188 non-null  float64
 17  cons.conf.idx              41188 non-null  float64
 18  euribor3m                  41188 non-null  float64
 19  nr.employed                41188 non-null  float64
 20  subbed_term_deposit        41188 non-null  object
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

In [58]:

```
# let's change string values to numerical
df.has_credit_in_default[df.has_credit_in_default == 'no'] = 0
df.has_credit_in_default[df.has_credit_in_default == 'yes'] = 1
df.head(20)
```

Out[58]:

	age	job	marital	education	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact_
0	56	housemaid	married	basic.4y	0	no	no	telephone	may	
1	57	services	married	high.school	NaN	no	no	telephone	may	
2	37	services	married	high.school	0	yes	no	telephone	may	
3	40	admin.	married	basic.6y	0	no	no	telephone	may	
4	56	services	married	high.school	0	no	yes	telephone	may	
5	45	services	married	basic.9y	NaN	no	no	telephone	may	
6	59	admin.	married	professional.course	0	no	no	telephone	may	
7	41	blue-collar	married	NaN	NaN	no	no	telephone	may	
8	24	technician	single	professional.course	0	yes	no	telephone	may	
9	25	services	single	high.school	0	yes	no	telephone	may	
10	41	blue-collar	married	NaN	NaN	no	no	telephone	may	
11	25	services	single	high.school	0	yes	no	telephone	may	
12	29	blue-collar	single	high.school	0	no	yes	telephone	may	
13	57	housemaid	divorced	basic.4y	0	yes	no	telephone	may	
14	35	blue-collar	married	basic.6y	0	yes	no	telephone	may	
15	54	retired	married	basic.9y	NaN	yes	yes	telephone	may	
16	35	blue-collar	married	basic.6y	0	yes	no	telephone	may	
17	46	blue-collar	married	basic.6y	NaN	yes	yes	telephone	may	
18	50	blue-collar	married	basic.9y	0	yes	yes	telephone	may	

	age	job	marital	education	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact_
19	39	management	single	basic.9y	NaN	no	no	telephone	may	

20 rows × 21 columns

In [59]:

```
# let's change string values to numerical
df.has_housing_loan[df.has_housing_loan == 'no'] = 0
df.has_housing_loan[df.has_housing_loan == 'yes'] = 1
df.head(20)
```

Out[59]:

	age	job	marital	education	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact_
0	56	housemaid	married	basic.4y	0	0	no	telephone	may	
1	57	services	married	high.school	NaN	0	no	telephone	may	
2	37	services	married	high.school	0	1	no	telephone	may	
3	40	admin.	married	basic.6y	0	0	no	telephone	may	
4	56	services	married	high.school	0	0	yes	telephone	may	
5	45	services	married	basic.9y	NaN	0	no	telephone	may	
6	59	admin.	married	professional.course	0	0	no	telephone	may	
7	41	blue-collar	married	NaN	NaN	0	no	telephone	may	
8	24	technician	single	professional.course	0	1	no	telephone	may	
9	25	services	single	high.school	0	1	no	telephone	may	
10	41	blue-collar	married	NaN	NaN	0	no	telephone	may	
11	25	services	single	high.school	0	1	no	telephone	may	
12	29	blue-collar	single	high.school	0	0	yes	telephone	may	
13	57	housemaid	divorced	basic.4y	0	1	no	telephone	may	
14	35	blue-collar	married	basic.6y	0	1	no	telephone	may	
15	54	retired	married	basic.9y	NaN	1	yes	telephone	may	
16	35	blue-collar	married	basic.6y	0	1	no	telephone	may	
17	46	blue-collar	married	basic.6y	NaN	1	yes	telephone	may	
18	50	blue-collar	married	basic.9y	0	1	yes	telephone	may	

	age	job	marital	education	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact_
19	39	management	single	basic.9y	NaN	0	no	telephone	may	

20 rows × 21 columns

In [60]:

```python
# let's change string values to numerical
df.has_personal_loan[df.has_personal_loan == 'no'] = 0
df.has_personal_loan[df.has_personal_loan == 'yes'] = 1
df.head(20)
```

Out[60]:

	age	job	marital	education	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact_
0	56	housemaid	married	basic.4y	0	0	0	telephone	may	
1	57	services	married	high.school	NaN	0	0	telephone	may	
2	37	services	married	high.school	0	1	0	telephone	may	
3	40	admin.	married	basic.6y	0	0	0	telephone	may	
4	56	services	married	high.school	0	0	1	telephone	may	
5	45	services	married	basic.9y	NaN	0	0	telephone	may	
6	59	admin.	married	professional.course	0	0	0	telephone	may	
7	41	blue-collar	married	NaN	NaN	0	0	telephone	may	
8	24	technician	single	professional.course	0	1	0	telephone	may	
9	25	services	single	high.school	0	1	0	telephone	may	
10	41	blue-collar	married	NaN	NaN	0	0	telephone	may	
11	25	services	single	high.school	0	1	0	telephone	may	
12	29	blue-collar	single	high.school	0	0	1	telephone	may	
13	57	housemaid	divorced	basic.4y	0	1	0	telephone	may	
14	35	blue-collar	married	basic.6y	0	1	0	telephone	may	
15	54	retired	married	basic.9y	NaN	1	1	telephone	may	
16	35	blue-collar	married	basic.6y	0	1	0	telephone	may	
17	46	blue-collar	married	basic.6y	NaN	1	1	telephone	may	
18	50	blue-collar	married	basic.9y	0	1	1	telephone	may	

	age	job	marital	education	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact_
19	39	management	single	basic.9y	NaN	0	0	telephone	may	

20 rows × 21 columns

In [61]:

```python
# let's change string values to numerical
df.subbed_term_deposit[df.subbed_term_deposit == 'no'] = 0
df.subbed_term_deposit[df.subbed_term_deposit == 'yes'] = 1
df.head(20)
```

Out[61]:

	age	job	marital	education	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact_
0	56	housemaid	married	basic.4y	0	0	0	telephone	may	
1	57	services	married	high.school	NaN	0	0	telephone	may	
2	37	services	married	high.school	0	1	0	telephone	may	
3	40	admin.	married	basic.6y	0	0	0	telephone	may	
4	56	services	married	high.school	0	0	1	telephone	may	
5	45	services	married	basic.9y	NaN	0	0	telephone	may	
6	59	admin.	married	professional.course	0	0	0	telephone	may	
7	41	blue-collar	married	NaN	NaN	0	0	telephone	may	
8	24	technician	single	professional.course	0	1	0	telephone	may	
9	25	services	single	high.school	0	1	0	telephone	may	
10	41	blue-collar	married	NaN	NaN	0	0	telephone	may	
11	25	services	single	high.school	0	1	0	telephone	may	
12	29	blue-collar	single	high.school	0	0	1	telephone	may	
13	57	housemaid	divorced	basic.4y	0	1	0	telephone	may	
14	35	blue-collar	married	basic.6y	0	1	0	telephone	may	
15	54	retired	married	basic.9y	NaN	1	1	telephone	may	
16	35	blue-collar	married	basic.6y	0	1	0	telephone	may	
17	46	blue-collar	married	basic.6y	NaN	1	1	telephone	may	
18	50	blue-collar	married	basic.9y	0	1	1	telephone	may	

	age	job	marital	education	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact_
19	39	management	single	basic.9y	NaN	0	0	telephone	may	

20 rows × 21 columns

In [62]:

```
# let's change string values to numerical
df.outcome_prev_campaign[df.outcome_prev_campaign == 'failure'] = 0
df.outcome_prev_campaign[df.outcome_prev_campaign == 'success'] = 1
df.outcome_prev_campaign[df.outcome_prev_campaign == 'nonexistent'] = np.nan
#df = df.replace('unknown', np.nan)
df
```

Out[62]:

	age	job	marital	education	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact
0	56	housemaid	married	basic.4y	0	0	0	telephone	may	
1	57	services	married	high.school	NaN	0	0	telephone	may	
2	37	services	married	high.school	0	1	0	telephone	may	
3	40	admin.	married	basic.6y	0	0	0	telephone	may	
4	56	services	married	high.school	0	0	1	telephone	may	
...	...	...	...	...	...	...	...	...	...	
41183	73	retired	married	professional.course	0	1	0	cellular	nov	
41184	46	blue-collar	married	professional.course	0	0	0	cellular	nov	
41185	56	retired	married	university.degree	0	1	0	cellular	nov	
41186	44	technician	married	professional.course	0	0	0	cellular	nov	
41187	74	retired	married	professional.course	0	1	0	cellular	nov	

41188 rows × 21 columns

In [63]:

```
# let's change string values to numerical
df.contact_type[df.contact_type == 'telephone'] = 0
df.contact_type[df.contact_type == 'cellular'] = 1
df
```

Out[63]:

	age	job	marital	education	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact

	age	job	marital	education	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact
0	56	housemaid	married	basic.4y	0	0	0	0	may	
1	57	services	married	high.school	NaN	0	0	0	may	
2	37	services	married	high.school	0	1	0	0	may	
3	40	admin.	married	basic.6y	0	0	0	0	may	
4	56	services	married	high.school	0	0	1	0	may	
...	...	...	...	...	...	...	...	...	...	
41183	73	retired	married	professional.course	0	1	0	1	nov	
41184	46	blue-collar	married	professional.course	0	0	0	1	nov	
41185	56	retired	married	university.degree	0	1	0	1	nov	
41186	44	technician	married	professional.course	0	0	0	1	nov	
41187	74	retired	married	professional.course	0	1	0	1	nov	

41188 rows × 21 columns

In [64]:

```
# let's change string values to numerical
df.marital[df.marital == 'divorced'] = 0
df.marital[df.marital == 'single'] = 1
df.marital[df.marital == 'married'] = 2
df
```

Out[64]:

	age	job	marital	education	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact
0	56	housemaid	2	basic.4y	0	0	0	0	may	
1	57	services	2	high.school	NaN	0	0	0	may	
2	37	services	2	high.school	0	1	0	0	may	
3	40	admin.	2	basic.6y	0	0	0	0	may	
4	56	services	2	high.school	0	0	1	0	may	
...	...	...	...	...	...	...	...	...	...	
41183	73	retired	2	professional.course	0	1	0	1	nov	
41184	46	blue-collar	2	professional.course	0	0	0	1	nov	
41185	56	retired	2	university.degree	0	1	0	1	nov	

	age	job	marital	education	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact
41186	44	technician	2	professional.course	0	0	0	1	nov	
41187	74	retired	2	professional.course	0	1	0	1	nov	

41188 rows × 21 columns

In [65]:

```
# let's change string values to numerical
df.last_contact_day_of_week[df.last_contact_day_of_week == 'mon'] = 1
df.last_contact_day_of_week[df.last_contact_day_of_week == 'tue'] = 2
df.last_contact_day_of_week[df.last_contact_day_of_week == 'wed'] = 3
df.last_contact_day_of_week[df.last_contact_day_of_week == 'thu'] = 4
df.last_contact_day_of_week[df.last_contact_day_of_week == 'fri'] = 5
df.last_contact_day_of_week[df.last_contact_day_of_week == 'sat'] = 6
df.last_contact_day_of_week[df.last_contact_day_of_week == 'sun'] = 7
df
```

Out[65]:

	age	job	marital	education	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact
0	56	housemaid	2	basic.4y	0	0	0	0	may	
1	57	services	2	high.school	NaN	0	0	0	may	
2	37	services	2	high.school	0	1	0	0	may	
3	40	admin.	2	basic.6y	0	0	0	0	may	
4	56	services	2	high.school	0	0	1	0	may	
...	...	...	...	...	...	...	...	...	...	
41183	73	retired	2	professional.course	0	1	0	1	nov	
41184	46	blue-collar	2	professional.course	0	0	0	1	nov	
41185	56	retired	2	university.degree	0	1	0	1	nov	
41186	44	technician	2	professional.course	0	0	0	1	nov	
41187	74	retired	2	professional.course	0	1	0	1	nov	

41188 rows × 21 columns

In [66]:

```
# let's change string values to numerical
df.last_contact_month[df.last_contact_month == 'jan'] = 1
df.last_contact_month[df.last_contact_month == 'feb'] = 2
```

```
df.last_contact_month[df.last_contact_month == 'mar'] = 3
df.last_contact_month[df.last_contact_month == 'apr'] = 4
df.last_contact_month[df.last_contact_month == 'may'] = 5
df.last_contact_month[df.last_contact_month == 'jun'] = 6
df.last_contact_month[df.last_contact_month == 'jul'] = 7
df.last_contact_month[df.last_contact_month == 'aug'] = 8
df.last_contact_month[df.last_contact_month == 'sep'] = 9
df.last_contact_month[df.last_contact_month == 'oct'] = 10
df.last_contact_month[df.last_contact_month == 'nov'] = 11
df.last_contact_month[df.last_contact_month == 'dec'] = 12
df
```

Out[66]:

	age	job	marital	education	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact
0	56	housemaid	2	basic.4y	0	0	0	0	5	
1	57	services	2	high.school	NaN	0	0	0	5	
2	37	services	2	high.school	0	1	0	0	5	
3	40	admin.	2	basic.6y	0	0	0	0	5	
4	56	services	2	high.school	0	0	1	0	5	
...	...	...	...	...	...	...	...	...	...	
41183	73	retired	2	professional.course	0	1	0	1	11	
41184	46	blue-collar	2	professional.course	0	0	0	1	11	
41185	56	retired	2	university.degree	0	1	0	1	11	
41186	44	technician	2	professional.course	0	0	0	1	11	
41187	74	retired	2	professional.course	0	1	0	1	11	

41188 rows × 21 columns

In [67]:

```
# let's get the dummy valaues for the categorical columns of 'job' and 'education'
df = pd.get_dummies(df, columns = ['job'])
df = pd.get_dummies(df, columns = ['education'])
df
```

Out[67]:

	age	marital	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact_day_of_week	last_contact_dura
0	56	2	0	0	0	0	5	1	
1	57	2	NaN	0	0	0	5	1	

	age	marital	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact_day_of_week	last_contact_dura
2	37	2	0	1	0	0	5	1	
3	40	2	0	0	0	0	5	1	
4	56	2	0	0	1	0	5	1	
...	...	...	...	...	...	...	...	...	
41183	73	2	0	1	0	1	11	5	
41184	46	2	0	0	0	1	11	5	
41185	56	2	0	1	0	1	11	5	
41186	44	2	0	0	0	1	11	5	
41187	74	2	0	1	0	1	11	5	

41188 rows × 37 columns

In [68]:

```
#since outcome_prev_campaign has mostly nonexistent values, i'm just going to remove the whole column
df = df.drop(columns=['outcome_prev_campaign'])
df = df.drop(columns=['num_days_prev_campaign'])
```

In [69]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 35 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   age                           41188 non-null  int64  
 1   marital                       41108 non-null  object 
 2   has_credit_in_default         32591 non-null  object 
 3   has_housing_loan              40198 non-null  object 
 4   has_personal_loan             40198 non-null  object 
 5   contact_type                  41188 non-null  object 
 6   last_contact_month            41188 non-null  object 
 7   last_contact_day_of_week      41188 non-null  object 
 8   last_contact_duration         41188 non-null  int64  
 9   num_contacts_this_campaign    41188 non-null  int64  
 10  num_contacts_prev_campaign    41188 non-null  int64  
 11  emp.var.rate                  41188 non-null  float64
 12  cons.price.idx                41188 non-null  float64
 13  cons.conf.idx                 41188 non-null  float64
```

```
 14  euribor3m                    41188 non-null  float64
 15  nr.employed                  41188 non-null  float64
 16  subbed_term_deposit          41188 non-null  object
 17  job_admin.                   41188 non-null  uint8
 18  job_blue-collar              41188 non-null  uint8
 19  job_entrepreneur             41188 non-null  uint8
 20  job_housemaid                41188 non-null  uint8
 21  job_management               41188 non-null  uint8
 22  job_retired                  41188 non-null  uint8
 23  job_self-employed            41188 non-null  uint8
 24  job_services                 41188 non-null  uint8
 25  job_student                  41188 non-null  uint8
 26  job_technician               41188 non-null  uint8
 27  job_unemployed               41188 non-null  uint8
 28  education_basic.4y           41188 non-null  uint8
 29  education_basic.6y           41188 non-null  uint8
 30  education_basic.9y           41188 non-null  uint8
 31  education_high.school        41188 non-null  uint8
 32  education_illiterate         41188 non-null  uint8
 33  education_professional.course  41188 non-null  uint8
 34  education_university.degree  41188 non-null  uint8
dtypes: float64(5), int64(4), object(8), uint8(18)
memory usage: 6.0+ MB
```

In [70]:

```
df['subbed_term_deposit'].value_counts()
```

Out[70]:

```
0    36548
1     4640
Name: subbed_term_deposit, dtype: int64
```

In [71]:

```
# now let's cast the cleaned data columns still marked as objects, as ints instead
df['subbed_term_deposit'] = df['subbed_term_deposit'].astype(int)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 35 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   age                          41188 non-null  int64
 1   marital                      41108 non-null  object
 2   has_credit_in_default        32591 non-null  object
 3   has_housing_loan             40198 non-null  object
 4   has_personal_loan            40198 non-null  object
 5   contact_type                 41188 non-null  object
 6   last_contact_month           41188 non-null  object
 7   last_contact_day_of_week     41188 non-null  object
 8   last_contact_duration        41188 non-null  int64
```

```
 9   num_contacts_this_campaign     41188 non-null  int64
 10  num_contacts_prev_campaign     41188 non-null  int64
 11  emp.var.rate                   41188 non-null  float64
 12  cons.price.idx                 41188 non-null  float64
 13  cons.conf.idx                  41188 non-null  float64
 14  euribor3m                      41188 non-null  float64
 15  nr.employed                    41188 non-null  float64
 16  subbed_term_deposit            41188 non-null  int32
 17  job_admin.                     41188 non-null  uint8
 18  job_blue-collar                41188 non-null  uint8
 19  job_entrepreneur               41188 non-null  uint8
 20  job_housemaid                  41188 non-null  uint8
 21  job_management                 41188 non-null  uint8
 22  job_retired                    41188 non-null  uint8
 23  job_self-employed              41188 non-null  uint8
 24  job_services                   41188 non-null  uint8
 25  job_student                    41188 non-null  uint8
 26  job_technician                 41188 non-null  uint8
 27  job_unemployed                 41188 non-null  uint8
 28  education_basic.4y             41188 non-null  uint8
 29  education_basic.6y             41188 non-null  uint8
 30  education_basic.9y             41188 non-null  uint8
 31  education_high.school          41188 non-null  uint8
 32  education_illiterate           41188 non-null  uint8
 33  education_professional.course  41188 non-null  uint8
 34  education_university.degree    41188 non-null  uint8
dtypes: float64(5), int32(1), int64(4), object(7), uint8(18)
memory usage: 5.9+ MB
```

In [ ]:

In [72]:

```
# saving the data to csv so I can look at it
df.to_csv('numerical.csv')
```

In [73]:

```
#Prepare the data for machine learning
##A little EDA goeas a long way
##Do you need to do anything about data types?
df
```

Out[73]:

	age	marital	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact_day_of_week	last_contact_dura
0	56	2	0	0	0	0	5	1	
1	57	2	NaN	0	0	0	5	1	
2	37	2	0	1	0	0	5	1	

	age	marital	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact_day_of_week	last_contact_dura
3	40	2	0	0	0	0	5	1	
4	56	2	0	0	1	0	5	1	
...	...	...	...	...	...	...	...	...	
41183	73	2	0	1	0	1	11	5	
41184	46	2	0	0	0	1	11	5	
41185	56	2	0	1	0	1	11	5	
41186	44	2	0	0	0	1	11	5	
41187	74	2	0	1	0	1	11	5	

41188 rows × 35 columns

In [74]:

```
# now let's cast the cleaned data columns still marked as objects, as ints instead
df['contact_type'] = df['contact_type'].astype(int)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 35 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   age                         41188 non-null  int64  
 1   marital                     41108 non-null  object 
 2   has_credit_in_default       32591 non-null  object 
 3   has_housing_loan            40198 non-null  object 
 4   has_personal_loan           40198 non-null  object 
 5   contact_type                41188 non-null  int32  
 6   last_contact_month          41188 non-null  object 
 7   last_contact_day_of_week    41188 non-null  object 
 8   last_contact_duration       41188 non-null  int64  
 9   num_contacts_this_campaign  41188 non-null  int64  
 10  num_contacts_prev_campaign  41188 non-null  int64  
 11  emp.var.rate                41188 non-null  float64
 12  cons.price.idx              41188 non-null  float64
 13  cons.conf.idx               41188 non-null  float64
 14  euribor3m                   41188 non-null  float64
 15  nr.employed                 41188 non-null  float64
 16  subbed_term_deposit         41188 non-null  int32  
 17  job_admin.                  41188 non-null  uint8  
 18  job_blue-collar             41188 non-null  uint8  
```

```
 19   job_entrepreneur               41188 non-null  uint8
 20   job_housemaid                  41188 non-null  uint8
 21   job_management                 41188 non-null  uint8
 22   job_retired                    41188 non-null  uint8
 23   job_self-employed              41188 non-null  uint8
 24   job_services                   41188 non-null  uint8
 25   job_student                    41188 non-null  uint8
 26   job_technician                 41188 non-null  uint8
 27   job_unemployed                 41188 non-null  uint8
 28   education_basic.4y             41188 non-null  uint8
 29   education_basic.6y             41188 non-null  uint8
 30   education_basic.9y             41188 non-null  uint8
 31   education_high.school          41188 non-null  uint8
 32   education_illiterate           41188 non-null  uint8
 33   education_professional.course  41188 non-null  uint8
 34   education_university.degree    41188 non-null  uint8
dtypes: float64(5), int32(2), int64(4), object(6), uint8(18)
memory usage: 5.7+ MB
```

In [75]:

```
# now let's cast the cleaned data columns still marked as objects, as ints instead
df['last_contact_month'] = df['last_contact_month'].astype(int)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 35 columns):
 #    Column                         Non-Null Count  Dtype
---   ------                         --------------  -----
 0    age                            41188 non-null  int64
 1    marital                        41108 non-null  object
 2    has_credit_in_default          32591 non-null  object
 3    has_housing_loan               40198 non-null  object
 4    has_personal_loan              40198 non-null  object
 5    contact_type                   41188 non-null  int32
 6    last_contact_month             41188 non-null  int32
 7    last_contact_day_of_week       41188 non-null  object
 8    last_contact_duration          41188 non-null  int64
 9    num_contacts_this_campaign     41188 non-null  int64
 10   num_contacts_prev_campaign     41188 non-null  int64
 11   emp.var.rate                   41188 non-null  float64
 12   cons.price.idx                 41188 non-null  float64
 13   cons.conf.idx                  41188 non-null  float64
 14   euribor3m                      41188 non-null  float64
 15   nr.employed                    41188 non-null  float64
 16   subbed_term_deposit            41188 non-null  int32
 17   job_admin.                     41188 non-null  uint8
 18   job_blue-collar                41188 non-null  uint8
 19   job_entrepreneur               41188 non-null  uint8
 20   job_housemaid                  41188 non-null  uint8
```

```
 21  job_management                 41188 non-null  uint8
 22  job_retired                    41188 non-null  uint8
 23  job_self-employed              41188 non-null  uint8
 24  job_services                   41188 non-null  uint8
 25  job_student                    41188 non-null  uint8
 26  job_technician                 41188 non-null  uint8
 27  job_unemployed                 41188 non-null  uint8
 28  education_basic.4y             41188 non-null  uint8
 29  education_basic.6y             41188 non-null  uint8
 30  education_basic.9y             41188 non-null  uint8
 31  education_high.school          41188 non-null  uint8
 32  education_illiterate           41188 non-null  uint8
 33  education_professional.course  41188 non-null  uint8
 34  education_university.degree    41188 non-null  uint8
dtypes: float64(5), int32(3), int64(4), object(5), uint8(18)
memory usage: 5.6+ MB
```

In [76]:

```python
# now let's cast the cleaned data columns still marked as objects, as ints instead
df['last_contact_day_of_week'] = df['last_contact_day_of_week'].astype(int)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 35 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   age                            41188 non-null  int64
 1   marital                        41108 non-null  object
 2   has_credit_in_default          32591 non-null  object
 3   has_housing_loan               40198 non-null  object
 4   has_personal_loan              40198 non-null  object
 5   contact_type                   41188 non-null  int32
 6   last_contact_month             41188 non-null  int32
 7   last_contact_day_of_week       41188 non-null  int32
 8   last_contact_duration          41188 non-null  int64
 9   num_contacts_this_campaign     41188 non-null  int64
 10  num_contacts_prev_campaign     41188 non-null  int64
 11  emp.var.rate                   41188 non-null  float64
 12  cons.price.idx                 41188 non-null  float64
 13  cons.conf.idx                  41188 non-null  float64
 14  euribor3m                      41188 non-null  float64
 15  nr.employed                    41188 non-null  float64
 16  subbed_term_deposit            41188 non-null  int32
 17  job_admin.                     41188 non-null  uint8
 18  job_blue-collar                41188 non-null  uint8
 19  job_entrepreneur               41188 non-null  uint8
 20  job_housemaid                  41188 non-null  uint8
 21  job_management                 41188 non-null  uint8
 22  job_retired                    41188 non-null  uint8
```

```
 23  job_self-employed              41188 non-null  uint8
 24  job_services                   41188 non-null  uint8
 25  job_student                    41188 non-null  uint8
 26  job_technician                 41188 non-null  uint8
 27  job_unemployed                 41188 non-null  uint8
 28  education_basic.4y             41188 non-null  uint8
 29  education_basic.6y             41188 non-null  uint8
 30  education_basic.9y             41188 non-null  uint8
 31  education_high.school          41188 non-null  uint8
 32  education_illiterate           41188 non-null  uint8
 33  education_professional.course  41188 non-null  uint8
 34  education_university.degree    41188 non-null  uint8
dtypes: float64(5), int32(4), int64(4), object(4), uint8(18)
memory usage: 5.4+ MB
```

In [77]:

```python
# since 31760 still seems like a very large dataset, I decided to drop all missing values
df = df.dropna()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 31760 entries, 0 to 41187
Data columns (total 35 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   age                            31760 non-null  int64
 1   marital                        31760 non-null  object
 2   has_credit_in_default          31760 non-null  object
 3   has_housing_loan               31760 non-null  object
 4   has_personal_loan              31760 non-null  object
 5   contact_type                   31760 non-null  int32
 6   last_contact_month             31760 non-null  int32
 7   last_contact_day_of_week       31760 non-null  int32
 8   last_contact_duration          31760 non-null  int64
 9   num_contacts_this_campaign     31760 non-null  int64
 10  num_contacts_prev_campaign     31760 non-null  int64
 11  emp.var.rate                   31760 non-null  float64
 12  cons.price.idx                 31760 non-null  float64
 13  cons.conf.idx                  31760 non-null  float64
 14  euribor3m                      31760 non-null  float64
 15  nr.employed                    31760 non-null  float64
 16  subbed_term_deposit            31760 non-null  int32
 17  job_admin.                     31760 non-null  uint8
 18  job_blue-collar                31760 non-null  uint8
 19  job_entrepreneur               31760 non-null  uint8
 20  job_housemaid                  31760 non-null  uint8
 21  job_management                 31760 non-null  uint8
 22  job_retired                    31760 non-null  uint8
 23  job_self-employed              31760 non-null  uint8
 24  job_services                   31760 non-null  uint8
```

```
 25  job_student                  31760 non-null  uint8
 26  job_technician               31760 non-null  uint8
 27  job_unemployed               31760 non-null  uint8
 28  education_basic.4y           31760 non-null  uint8
 29  education_basic.6y           31760 non-null  uint8
 30  education_basic.9y           31760 non-null  uint8
 31  education_high.school        31760 non-null  uint8
 32  education_illiterate         31760 non-null  uint8
 33  education_professional.course 31760 non-null  uint8
 34  education_university.degree  31760 non-null  uint8
dtypes: float64(5), int32(4), int64(4), object(4), uint8(18)
memory usage: 4.4+ MB
```

In [78]:

```python
# now let's cast the cleaned data columns still marked as objects, as ints instead
df['marital'] = df['marital'].astype(int)
df['has_credit_in_default'] = df['has_credit_in_default'].astype(int)
df['has_housing_loan'] = df['has_housing_loan'].astype(int)
df['has_personal_loan'] = df['has_personal_loan'].astype(int)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 31760 entries, 0 to 41187
Data columns (total 35 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   age                           31760 non-null  int64
 1   marital                       31760 non-null  int32
 2   has_credit_in_default         31760 non-null  int32
 3   has_housing_loan              31760 non-null  int32
 4   has_personal_loan             31760 non-null  int32
 5   contact_type                  31760 non-null  int32
 6   last_contact_month            31760 non-null  int32
 7   last_contact_day_of_week      31760 non-null  int32
 8   last_contact_duration         31760 non-null  int64
 9   num_contacts_this_campaign    31760 non-null  int64
 10  num_contacts_prev_campaign    31760 non-null  int64
 11  emp.var.rate                  31760 non-null  float64
 12  cons.price.idx                31760 non-null  float64
 13  cons.conf.idx                 31760 non-null  float64
 14  euribor3m                     31760 non-null  float64
 15  nr.employed                   31760 non-null  float64
 16  subbed_term_deposit           31760 non-null  int32
 17  job_admin.                    31760 non-null  uint8
 18  job_blue-collar               31760 non-null  uint8
 19  job_entrepreneur              31760 non-null  uint8
 20  job_housemaid                 31760 non-null  uint8
 21  job_management                31760 non-null  uint8
 22  job_retired                   31760 non-null  uint8
 23  job_self-employed             31760 non-null  uint8
```

```
 24  job_services                 31760 non-null  uint8
 25  job_student                  31760 non-null  uint8
 26  job_technician               31760 non-null  uint8
 27  job_unemployed               31760 non-null  uint8
 28  education_basic.4y           31760 non-null  uint8
 29  education_basic.6y           31760 non-null  uint8
 30  education_basic.9y           31760 non-null  uint8
 31  education_high.school        31760 non-null  uint8
 32  education_illiterate         31760 non-null  uint8
 33  education_professional.course 31760 non-null  uint8
 34  education_university.degree  31760 non-null  uint8
dtypes: float64(5), int32(8), int64(4), uint8(18)
memory usage: 3.9 MB
```

In [79]:

```
df.describe()
```

Out[79]:

	age	marital	has_credit_in_default	has_housing_loan	has_personal_loan	contact_type	last_contact_month	last_contact_day_of_week	las
count	31760.00000	31760.000000	31760.000000	31760.000000	31760.000000	31760.000000	31760.000000	31760.000000	
mean	39.14745	1.458249	0.000094	0.540932	0.156360	0.669395	6.700346	2.982588	
std	10.47959	0.692782	0.009719	0.498330	0.363202	0.470438	2.133558	1.408518	
min	17.00000	0.000000	0.000000	0.000000	0.000000	0.000000	3.000000	1.000000	
25%	31.00000	1.000000	0.000000	0.000000	0.000000	0.000000	5.000000	2.000000	
50%	37.00000	2.000000	0.000000	1.000000	0.000000	1.000000	6.000000	3.000000	
75%	46.00000	2.000000	0.000000	1.000000	0.000000	1.000000	8.000000	4.000000	
max	95.00000	2.000000	1.000000	1.000000	1.000000	1.000000	12.000000	5.000000	

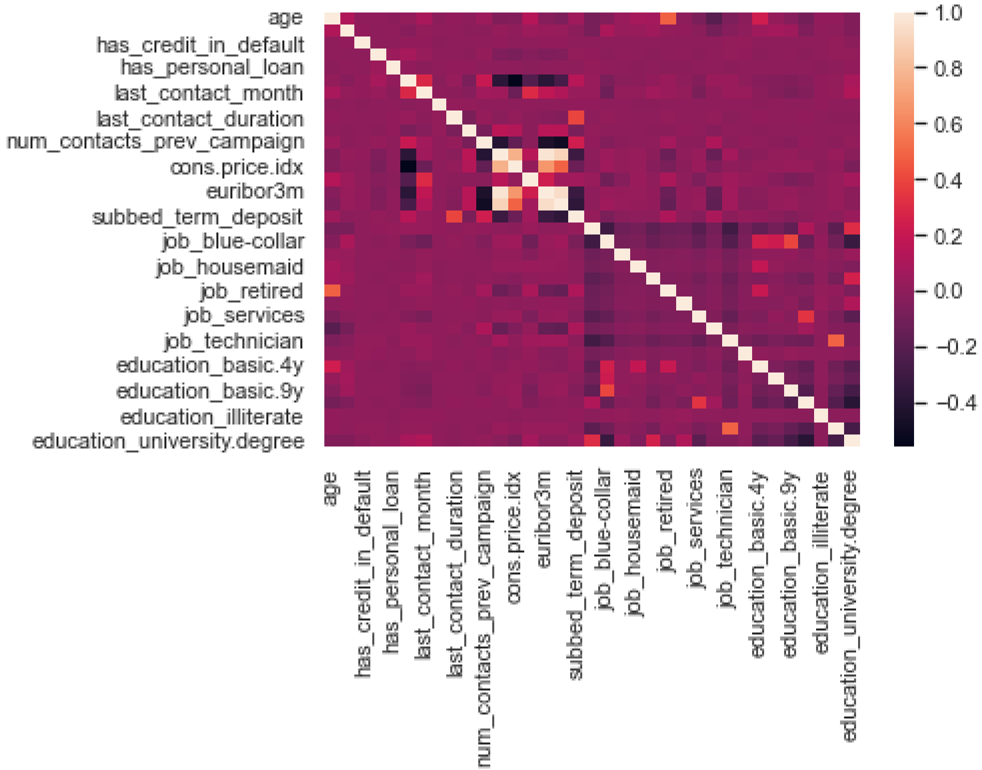
8 rows × 35 columns

In [80]:

```
# Let's see the correlation heatmap, just out of curiosity
sns.heatmap(df.corr())
```

Out[80]:

```
<AxesSubplot:>
```



In [81]:

```
df.shape
```

Out[81]: (31760, 35)

KNN Analysis

- What is your objective from the analysis?
- Answer: The objective from the KNN analysis is to predict if the client has subscribed a term deposit or not. That is to say, the KNN analysis will predict the subbed_term_deposit column in the dataframe

In [82]:

```
##What is your optimal K?
##How about accuracy rate?
###Discover any insights from this analysis?
##Include numbers/graphs corresponding to your conclusions
##Discuss ways to improve the performance of your KNN model
##Defend and backup your thoughts!!!!!!
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.metrics import r2_score
from sklearn.metrics import explained_variance_score
```

In [83]:

```
# first, I will define my features and target(s), and create my train/test data
cols = df.columns
target_col = 'subbed_term_deposit'
feat_cols = [c for c in cols if c != target_col]

array = df.values

X = np.delete(array, 17, 1) #index 17 is the subbed_term_deposit column
y = array[:, 17]
#print(A,B)
print(X)
```

```
[[56.  2.  0. ...  0.  0.  0.]
 [37.  2.  0. ...  0.  0.  0.]
 [40.  2.  0. ...  0.  0.  0.]
 ...
 [56.  2.  0. ...  0.  0.  1.]
 [44.  2.  0. ...  0.  1.  0.]
 [74.  2.  0. ...  0.  1.  0.]]
```

In [84]:

```
# after defining features and targets, it's time to create train/test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [85]:

```
# after creating the train/test data, I will iterate on the value of the clusters
# This will tell me what is the ideal value of k
# I will set the range of k values to test as 0-39, inclusive.
scores = []
num_clusters = 40
print(f'Features: {feat_cols} \nTarget: {target_col}')

# remember the ending number for range is not inclusive
for k in range(2, num_clusters):
    # output to let us know where we are
    # n_jobs=-1 will use all processors on your system
    model = KNeighborsRegressor(n_neighbors=k, n_jobs=-1)
    model.fit(X_train, y_train)
    scores.append(model.score(X_test, y_test))

    preds = model.predict(X_test)
    differs = y_test - preds

    print(f'Evaluating {k} clusters: score = {model.score(X_test, y_test)}, \nr2_score = {r2_score(y_test,preds)}, explained_varian
```

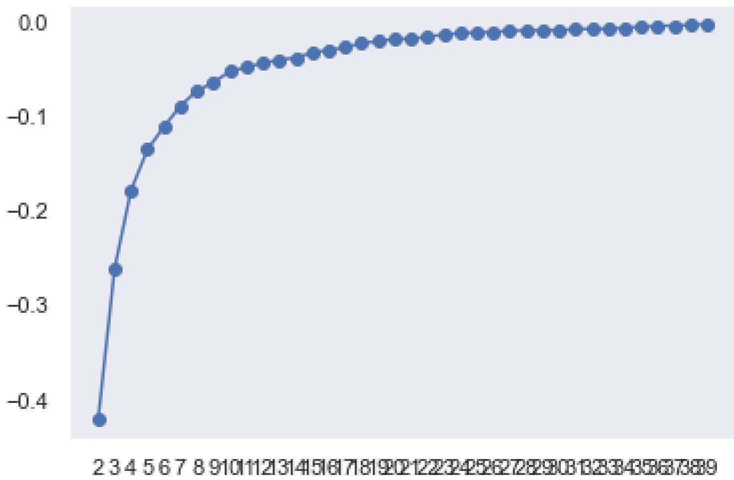
```
Features: ['age', 'marital', 'has_credit_in_default', 'has_housing_loan', 'has_personal_loan', 'contact_type', 'last_contact_month',
'last_contact_day_of_week', 'last_contact_duration', 'num_contacts_this_campaign', 'num_contacts_prev_campaign', 'emp.var.rate', 'co
```

```
ns.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed', 'job_admin.', 'job_blue-collar', 'job_entrepreneur', 'job_housemaid', 'j
ob_management', 'job_retired', 'job_self-employed', 'job_services', 'job_student', 'job_technician', 'job_unemployed', 'education_ba
sic.4y', 'education_basic.6y', 'education_basic.9y', 'education_high.school', 'education_illiterate', 'education_professional.cours
e', 'education_university.degree']
Target: subbed_term_deposit
Evaluating 2 clusters: score = -0.42126379276254045,
r2_score = -0.42126379276254045, explained_variance_score = -0.4132683495007585
Evaluating 3 clusters: score = -0.26123532809860683,
r2_score = -0.26123532809860683, explained_variance_score = -0.2542704086350096
Evaluating 4 clusters: score = -0.1785173181356421,
r2_score = -0.1785173181356421, explained_variance_score = -0.17239580688834022
Evaluating 5 clusters: score = -0.1353439244902368,
r2_score = -0.1353439244902368, explained_variance_score = -0.13016606435988543
Evaluating 6 clusters: score = -0.11140647986380303,
r2_score = -0.11140647986380303, explained_variance_score = -0.10627816834927506
Evaluating 7 clusters: score = -0.08921998135399267,
r2_score = -0.08921998135399267, explained_variance_score = -0.08458168862037296
Evaluating 8 clusters: score = -0.0733914762816339,
r2_score = -0.0733914762816339, explained_variance_score = -0.06929873868421943
Evaluating 9 clusters: score = -0.06453123117897075,
r2_score = -0.06453123117897075, explained_variance_score = -0.060740070803476875
Evaluating 10 clusters: score = -0.05287474999705166,
r2_score = -0.05287474999705166, explained_variance_score = -0.04938081659499671
Evaluating 11 clusters: score = -0.04860137522310515,
r2_score = -0.04860137522310515, explained_variance_score = -0.04534404617176402
Evaluating 12 clusters: score = -0.043998885510512675,
r2_score = -0.043998885510512675, explained_variance_score = -0.04105918470300174
Evaluating 13 clusters: score = -0.04068553199710423,
r2_score = -0.04068553199710423, explained_variance_score = -0.03776819711059631
Evaluating 14 clusters: score = -0.038350201913248894,
r2_score = -0.038350201913248894, explained_variance_score = -0.035548869783537995
Evaluating 15 clusters: score = -0.03297963611172605,
r2_score = -0.03297963611172605, explained_variance_score = -0.03034178557381506
Evaluating 16 clusters: score = -0.03036184447124235,
r2_score = -0.03036184447124235, explained_variance_score = -0.02773043556838295
Evaluating 17 clusters: score = -0.027564196764883864,
r2_score = -0.027564196764883864, explained_variance_score = -0.024963936899336403
Evaluating 18 clusters: score = -0.023169376687789844,
r2_score = -0.023169376687789844, explained_variance_score = -0.020730138281500388
Evaluating 19 clusters: score = -0.02079269390295857,
r2_score = -0.02079269390295857, explained_variance_score = -0.018411687330368576
Evaluating 20 clusters: score = -0.018704276404909903,
r2_score = -0.018704276404909903, explained_variance_score = -0.016404510072293155
Evaluating 21 clusters: score = -0.01833468087408896,
r2_score = -0.01833468087408896, explained_variance_score = -0.0160668293229691
Evaluating 22 clusters: score = -0.016348975979568303,
r2_score = -0.016348975979568303, explained_variance_score = -0.01415125308633014
Evaluating 23 clusters: score = -0.013838947671744206,
r2_score = -0.013838947671744206, explained_variance_score = -0.0116815441359015
Evaluating 24 clusters: score = -0.012513431634096905,
```

```
r2_score = -0.012513431634096905, explained_variance_score = -0.010452731254430692
Evaluating 25 clusters: score = -0.011887485863320002,
r2_score = -0.011887485863320002, explained_variance_score = -0.009853120495814993
Evaluating 26 clusters: score = -0.011220638229368252,
r2_score = -0.011220638229368252, explained_variance_score = -0.009157335047467852
Evaluating 27 clusters: score = -0.009893013469356449,
r2_score = -0.009893013469356449, explained_variance_score = -0.007792361208622589
Evaluating 28 clusters: score = -0.009479401644376662,
r2_score = -0.009479401644376662, explained_variance_score = -0.007377037151362398
Evaluating 29 clusters: score = -0.009438335347843996,
r2_score = -0.009438335347843996, explained_variance_score = -0.007339583438134234
Evaluating 30 clusters: score = -0.0091588422708897,
r2_score = -0.0091588422708897, explained_variance_score = -0.0070942233642246055
Evaluating 31 clusters: score = -0.0083764099277992,
r2_score = -0.0083764099277992, explained_variance_score = -0.006342099338464191
Evaluating 32 clusters: score = -0.007731617252954015,
r2_score = -0.007731617252954015, explained_variance_score = -0.0057030585407331635
Evaluating 33 clusters: score = -0.008077592821647883,
r2_score = -0.008077592821647883, explained_variance_score = -0.0060589176943954826
Evaluating 34 clusters: score = -0.006518588620151977,
r2_score = -0.006518588620151977, explained_variance_score = -0.0045474787173231945
Evaluating 35 clusters: score = -0.00604752859358304,
r2_score = -0.00604752859358304, explained_variance_score = -0.004079209431235631
Evaluating 36 clusters: score = -0.004722872118584132,
r2_score = -0.004722872118584132, explained_variance_score = -0.002749064182495209
Evaluating 37 clusters: score = -0.004440118866176013,
r2_score = -0.004440118866176013, explained_variance_score = -0.0024797403146314956
Evaluating 38 clusters: score = -0.003867458779169386,
r2_score = -0.003867458779169386, explained_variance_score = -0.0019464503262627275
Evaluating 39 clusters: score = -0.0027128934943549954,
r2_score = -0.0027128934943549954, explained_variance_score = -0.0008314849775479249
```

In [86]:

```
# display the resutls
plt.plot(range(2, num_clusters), scores)
plt.scatter(range(2, num_clusters), scores)
plt.grid()
_ =plt.xticks(range(2, num_clusters))
```



So, we can see that the optimal value of k is the maximum value tested, or 39 in this case. It appears that as the value of k increases for this specific dataset, the score of the model approaches 0. It is said that a negative score indicates poor performance. Why is my score negative?? What is wrong with the model. Is it the train/test data??

In [87]:

```
# After learning the optimal value of k, we re-run the model
# This run of the model is using the optimal value of k
model = KNeighborsRegressor(n_neighbors=num_clusters, n_jobs=-1)
model.fit(X_train, y_train)

# gather the predictations that our model made for our test set
preds = model.predict(X_test)

# display the actuals and predictions for the test set
print('Actuals for test data set')
print(y_test)
print('Predictions for test data set')
print(preds)
```

```
Actuals for test data set
[0. 0. 0. ... 0. 1. 0.]
Predictions for test data set
[0.325 0.3   0.25  ... 0.175 0.225 0.325]
```

In [88]:

```
differs = y_test - preds

print(f'Differences between the two sets:\n{differs}\n')

print(f'r2_score: {r2_score(y_test,preds)}')
print(f'explained_variance_score = {explained_variance_score(y_test,preds)}')
```

```
Differences between the two sets:
[-0.325 -0.3   -0.25  ... -0.175  0.775 -0.325]

r2_score: -0.002923447451305794
explained_variance_score = -0.0010620582884426355
```

Discover any insights from this analysis?

- From the analysis, it can be said that the model was really not so accurate in its performance. Its top score only reached a value of -0.001, a value close to 0 but still negative. This indicates the model is not very accurate. The test dataset created used a proportion of 0.3.

Include numbers/graphs corresponding to your conclusions

- As seen above, the scores of the model with regard to the k value used were plotted to clearly indicate the highest scoring k value. For this specific model, it was found that as the k value increased, the score of the model did as well. Therefore, it was pertinent to use the maximum k value tested.

Discuss ways to improve the performance of your KNN model

- I believe the proportion of the test data created could be adjusted to hopefully increase the score and make it positive at least.
- The model could also be improved if the data was cleaned more thoroughly. After initially cleaning the data, I had the choice to impute the remaining missing data or drop the remaining missing data. I saw that there were 31760 datapoints remaining if I chose to remove the missing data. I then decided that 31760 datapoints was still a very large dataset so I went forward with removing the missing data.

Defend and backup your thoughts!!!!!!

Part 2: Comparison to other supervised algorithm

As we saw in the lecture notebook, algorithm performance varies based on the algorithm used. The lecture demostrated using K-Fold Cross-Validation to compare the performance of several algorithm for the same dataset.

- At the end of part 1 you discussed ways to improve the performance of you KNN model.
  - Implement one of those methods to improve your KNN model performance.
  - Rerun a KNN analysis for your improved dataset
  - Discuss the change in performance from the model in part 1

Let's try test_size = 0.2, or 20%

In [89]:

```
# I will choose to find a better proportion of test data in order to
# improve the model score
# The former test proportion was 0.3. I will try
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [90]:

```
# after creating the train/test data, I will iterate on the value of the clusters
```

```python
# This will tell me what is the ideal value of k
# I will set the range of k values to test as 0-39, inclusive.
scores = []
num_clusters = 40
print(f'Features: {feat_cols} \nTarget: {target_col}')

# remember the ending number for range is not inclusive
for k in range(2, num_clusters):
    # output to let us know where we are
    # n_jobs=-1 will use all processors on your system
    model = KNeighborsRegressor(n_neighbors=k, n_jobs=-1)
    model.fit(X_train, y_train)
    scores.append(model.score(X_test, y_test))

    preds = model.predict(X_test)
    differs = y_test - preds

    print(f'Evaluating {k} clusters: score = {model.score(X_test, y_test)}, \nr2_score = {r2_score(y_test,preds)}, explained_varian
```

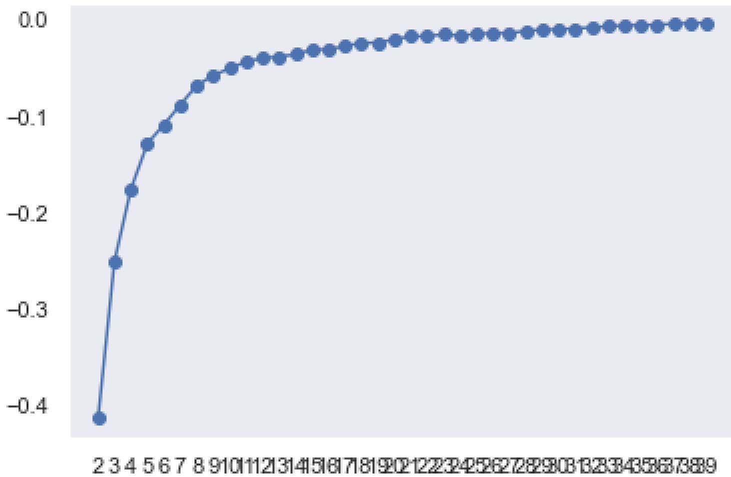
```
Features: ['age', 'marital', 'has_credit_in_default', 'has_housing_loan', 'has_personal_loan', 'contact_type', 'last_contact_month',
'last_contact_day_of_week', 'last_contact_duration', 'num_contacts_this_campaign', 'num_contacts_prev_campaign', 'emp.var.rate', 'co
ns.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed', 'job_admin.', 'job_blue-collar', 'job_entrepreneur', 'job_housemaid', 'j
ob_management', 'job_retired', 'job_self-employed', 'job_services', 'job_student', 'job_technician', 'job_unemployed', 'education_ba
sic.4y', 'education_basic.6y', 'education_basic.9y', 'education_high.school', 'education_illiterate', 'education_professional.cours
e', 'education_university.degree']
Target: subbed_term_deposit
Evaluating 2 clusters: score = -0.4141622247609509,
r2_score = -0.4141622247609509, explained_variance_score = -0.407452604892095
Evaluating 3 clusters: score = -0.2504943252459175,
r2_score = -0.2504943252459175, explained_variance_score = -0.24375569083849347
Evaluating 4 clusters: score = -0.1758342384074867,
r2_score = -0.1758342384074867, explained_variance_score = -0.16922567645816033
Evaluating 5 clusters: score = -0.12856229236764283,
r2_score = -0.12856229236764283, explained_variance_score = -0.12278579633484088
Evaluating 6 clusters: score = -0.10920401814669334,
r2_score = -0.10920401814669334, explained_variance_score = -0.10373526276710976
Evaluating 7 clusters: score = -0.0883632875989564,
r2_score = -0.0883632875989564, explained_variance_score = -0.08293184876260384
Evaluating 8 clusters: score = -0.0684126825145932,
r2_score = -0.0684126825145932, explained_variance_score = -0.06363371688132835
Evaluating 9 clusters: score = -0.0578560710874938,
r2_score = -0.0578560710874938, explained_variance_score = -0.05332056627173376
Evaluating 10 clusters: score = -0.05061934865960338,
r2_score = -0.05061934865960338, explained_variance_score = -0.04635759182459842
Evaluating 11 clusters: score = -0.04365718678715469,
r2_score = -0.04365718678715469, explained_variance_score = -0.03965387629418804
Evaluating 12 clusters: score = -0.04008310336972776,
r2_score = -0.04008310336972776, explained_variance_score = -0.03638643275585962
Evaluating 13 clusters: score = -0.0384101137418138,
```

```
r2_score = -0.0384101137418138, explained_variance_score = -0.034840853043694775
Evaluating 14 clusters: score = -0.03617264873679771,
r2_score = -0.03617264873679771, explained_variance_score = -0.032819984727172846
Evaluating 15 clusters: score = -0.031155924053758666,
r2_score = -0.031155924053758666, explained_variance_score = -0.02806468513678717
Evaluating 16 clusters: score = -0.031007738775254534,
r2_score = -0.031007738775254534, explained_variance_score = -0.027904520708461833
Evaluating 17 clusters: score = -0.027549275941579632,
r2_score = -0.027549275941579632, explained_variance_score = -0.0244817529133754
Evaluating 18 clusters: score = -0.02423200193051578,
r2_score = -0.02423200193051578, explained_variance_score = -0.021226285926922994
Evaluating 19 clusters: score = -0.023977843150064793,
r2_score = -0.023977843150064793, explained_variance_score = -0.02101068657562477
Evaluating 20 clusters: score = -0.02053485156262247,
r2_score = -0.02053485156262247, explained_variance_score = -0.017561808479005903
Evaluating 21 clusters: score = -0.017441327599653222,
r2_score = -0.017441327599653222, explained_variance_score = -0.014640455943458752
Evaluating 22 clusters: score = -0.01725274868823834,
r2_score = -0.01725274868823834, explained_variance_score = -0.01456739412753505
Evaluating 23 clusters: score = -0.015259529696720397,
r2_score = -0.015259529696720397, explained_variance_score = -0.012597172209179108
Evaluating 24 clusters: score = -0.016562331874433678,
r2_score = -0.016562331874433678, explained_variance_score = -0.013966234908910025
Evaluating 25 clusters: score = -0.015036776512934447,
r2_score = -0.015036776512934447, explained_variance_score = -0.012601098909216013
Evaluating 26 clusters: score = -0.01385367422016115,
r2_score = -0.01385367422016115, explained_variance_score = -0.01146409251063818
Evaluating 27 clusters: score = -0.014253406036373706,
r2_score = -0.014253406036373706, explained_variance_score = -0.011991667835027142
Evaluating 28 clusters: score = -0.011591762638168523,
r2_score = -0.011591762638168523, explained_variance_score = -0.009301578128238841
Evaluating 29 clusters: score = -0.010686852257684842,
r2_score = -0.010686852257684842, explained_variance_score = -0.008436489965333971
Evaluating 30 clusters: score = -0.009958843522155592,
r2_score = -0.009958843522155592, explained_variance_score = -0.007657922241871118
Evaluating 31 clusters: score = -0.009308531988542867,
r2_score = -0.009308531988542867, explained_variance_score = -0.006957591942500008
Evaluating 32 clusters: score = -0.008113869152881925,
r2_score = -0.008113869152881925, explained_variance_score = -0.005788627242797162
Evaluating 33 clusters: score = -0.007052818130356631,
r2_score = -0.007052818130356631, explained_variance_score = -0.0047536452118226435
Evaluating 34 clusters: score = -0.006111099528770758,
r2_score = -0.006111099528770758, explained_variance_score = -0.0037783869667589176
Evaluating 35 clusters: score = -0.005508751559779812,
r2_score = -0.005508751559779812, explained_variance_score = -0.003208638428437993
Evaluating 36 clusters: score = -0.0055975079499497316,
r2_score = -0.0055975079499497316, explained_variance_score = -0.0032894148732793838
Evaluating 37 clusters: score = -0.0041863371147565775,
r2_score = -0.0041863371147565775, explained_variance_score = -0.0018918028863823544
Evaluating 38 clusters: score = -0.003509939746932033 7,
```

```
r2_score = -0.0035099397469320337, explained_variance_score = -0.0012112809259798851
Evaluating 39 clusters: score = -0.003479020984165526,
r2_score = -0.003479020984165526, explained_variance_score = -0.001201626840039438
```

In [91]:

```python
# display the resutls
plt.plot(range(2, num_clusters), scores)
plt.scatter(range(2, num_clusters), scores)
plt.grid()
_ =plt.xticks(range(2, num_clusters))
```



In [92]:

```python
# After learning the optimal value of k, we re-run the model
# This run of the model is using the optimal value of k
model = KNeighborsRegressor(n_neighbors=num_clusters, n_jobs=-1)
model.fit(X_train, y_train)

# gather the predictations that our model made for our test set
preds = model.predict(X_test)

# display the actuals and predictions for the test set
print('Actuals for test data set')
print(y_test)
print('Predictions for test data set')
print(preds)
```

```
Actuals for test data set
[0. 0. 0. ... 0. 0. 0.]
Predictions for test data set
[0.325 0.3   0.325 ... 0.3   0.125 0.425]
```

In [93]:

```python
differs = y_test - preds
```

```
print(f'Differences between the two sets:\n{differs}\n')

print(f'r2_score: {r2_score(y_test,preds)}')
print(f'explained_variance_score = {explained_variance_score(y_test,preds)}')
```

```
Differences between the two sets:
[-0.325 -0.3   -0.325 ... -0.3   -0.125 -0.425]

r2_score: -0.0033444204134649436
explained_variance_score = -0.0010440648523450946
```

So, for test_size = 0.2, score = -0.001

Now, let's try test_size = 0.15

In [94]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=42)
```

In [95]:

```
# after creating the train/test data, I will iterate on the value of the clusters
# This will tell me what is the ideal value of k
# I will set the range of k values to test as 0-39, inclusive.
scores = []
num_clusters = 40
print(f'Features: {feat_cols} \nTarget: {target_col}')

# remember the ending number for range is not inclusive
for k in range(2, num_clusters):
    # output to let us know where we are
    # n_jobs=-1 will use all processors on your system
    model = KNeighborsRegressor(n_neighbors=k, n_jobs=-1)
    model.fit(X_train, y_train)
    scores.append(model.score(X_test, y_test))

    preds = model.predict(X_test)
    differs = y_test - preds

    print(f'Evaluating {k} clusters: score = {model.score(X_test, y_test)}, \nr2_score = {r2_score(y_test,preds)}, explained_varian
```

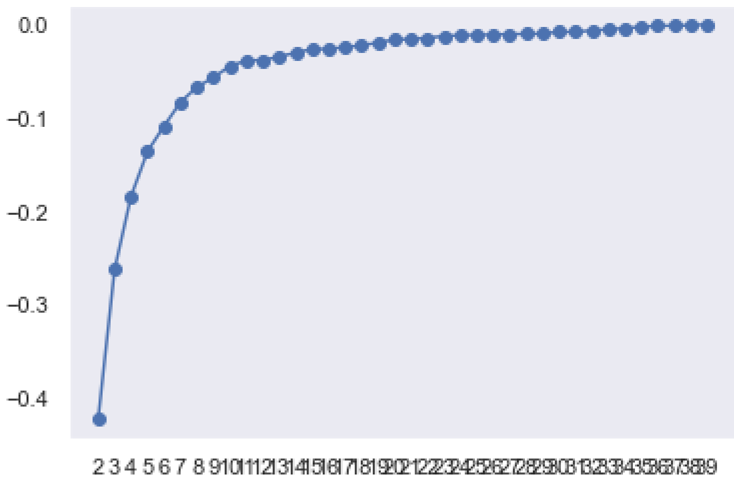
```
Features: ['age', 'marital', 'has_credit_in_default', 'has_housing_loan', 'has_personal_loan', 'contact_type', 'last_contact_month',
'last_contact_day_of_week', 'last_contact_duration', 'num_contacts_this_campaign', 'num_contacts_prev_campaign', 'emp.var.rate', 'co
ns.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed', 'job_admin.', 'job_blue-collar', 'job_entrepreneur', 'job_housemaid', 'j
ob_management', 'job_retired', 'job_self-employed', 'job_services', 'job_student', 'job_technician', 'job_unemployed', 'education_ba
sic.4y', 'education_basic.6y', 'education_basic.9y', 'education_high.school', 'education_illiterate', 'education_professional.cours
e', 'education_university.degree']
Target: subbed_term_deposit
Evaluating 2 clusters: score = -0.4224716222816851,
r2_score = -0.4224716222816851, explained_variance_score = -0.41619077356934486
Evaluating 3 clusters: score = -0.2619601714992257,
r2_score = -0.2619601714992257, explained_variance_score = -0.25588877228294327
```

```
Evaluating 4 clusters: score = -0.18383415758792032,
r2_score = -0.18383415758792032, explained_variance_score = -0.17818868561257628
Evaluating 5 clusters: score = -0.13591169724770635,
r2_score = -0.13591169724770635, explained_variance_score = -0.13096082512175755
Evaluating 6 clusters: score = -0.1089136887907276,
r2_score = -0.1089136887907276, explained_variance_score = -0.10430527141459311
Evaluating 7 clusters: score = -0.08336491770510412,
r2_score = -0.08336491770510412, explained_variance_score = -0.07880502560988689
Evaluating 8 clusters: score = -0.0671171861726978,
r2_score = -0.0671171861726978, explained_variance_score = -0.06314501350308621
Evaluating 9 clusters: score = -0.05698743592244382,
r2_score = -0.05698743592244382, explained_variance_score = -0.05321684657235748
Evaluating 10 clusters: score = -0.0446774921423716,
r2_score = -0.0446774921423716, explained_variance_score = -0.041148706942320645
Evaluating 11 clusters: score = -0.0388100054619025,
r2_score = -0.0388100054619025, explained_variance_score = -0.03547803848184339
Evaluating 12 clusters: score = -0.038479519434061915,
r2_score = -0.038479519434061915, explained_variance_score = -0.03524959743419709
Evaluating 13 clusters: score = -0.034664841932616985,
r2_score = -0.034664841932616985, explained_variance_score = -0.031580599007289045
Evaluating 14 clusters: score = -0.030402179029103804,
r2_score = -0.030402179029103804, explained_variance_score = -0.027505586045580177
Evaluating 15 clusters: score = -0.026566015875712612,
r2_score = -0.026566015875712612, explained_variance_score = -0.023944162663130042
Evaluating 16 clusters: score = -0.026530538980870766,
r2_score = -0.026530538980870766, explained_variance_score = -0.024036143479307492
Evaluating 17 clusters: score = -0.02440573241648969,
r2_score = -0.02440573241648969, explained_variance_score = -0.021892474459952238
Evaluating 18 clusters: score = -0.022850250765050317,
r2_score = -0.022850250765050317, explained_variance_score = -0.020485604087181475
Evaluating 19 clusters: score = -0.019578646817228096,
r2_score = -0.019578646817228096, explained_variance_score = -0.017260448791543537
Evaluating 20 clusters: score = -0.016433565876656342,
r2_score = -0.016433565876656342, explained_variance_score = -0.014230235587268991
Evaluating 21 clusters: score = -0.01594780227188819,
r2_score = -0.01594780227188819, explained_variance_score = -0.013827361465218457
Evaluating 22 clusters: score = -0.014692751289654638,
r2_score = -0.014692751289654638, explained_variance_score = -0.012684788237766087
Evaluating 23 clusters: score = -0.012690944823094075,
r2_score = -0.012690944823094075, explained_variance_score = -0.010725406997818654
Evaluating 24 clusters: score = -0.011814214358104813,
r2_score = -0.011814214358104813, explained_variance_score = -0.009999053498433552
Evaluating 25 clusters: score = -0.011473173632347677,
r2_score = -0.011473173632347677, explained_variance_score = -0.009787987031373513
Evaluating 26 clusters: score = -0.011159936192740805,
r2_score = -0.011159936192740805, explained_variance_score = -0.009484063848454793
Evaluating 27 clusters: score = -0.01068764749257367,
r2_score = -0.01068764749257367, explained_variance_score = -0.009062780924363523
Evaluating 28 clusters: score = -0.010300534507253323,
r2_score = -0.010300534507253323, explained_variance_score = -0.00868608372354207
```

```
Evaluating 29 clusters: score = -0.01012123858059355,
r2_score = -0.01012123858059355, explained_variance_score = -0.008474604358555515
Evaluating 30 clusters: score = -0.008043527249216531,
r2_score = -0.008043527249216531, explained_variance_score = -0.006450244656183157
Evaluating 31 clusters: score = -0.008227717656104083,
r2_score = -0.008227717656104083, explained_variance_score = -0.006629424743543
Evaluating 32 clusters: score = -0.006629332044356895,
r2_score = -0.006629332044356895, explained_variance_score = -0.005043940361040944
Evaluating 33 clusters: score = -0.005429465027596558,
r2_score = -0.005429465027596558, explained_variance_score = -0.0038128864671356233
Evaluating 34 clusters: score = -0.004304562329443762,
r2_score = -0.004304562329443762, explained_variance_score = -0.002671796879010646
Evaluating 35 clusters: score = -0.0028563930669105453,
r2_score = -0.0028563930669105453, explained_variance_score = -0.0012505699272571391
Evaluating 36 clusters: score = -0.001711705549079623,
r2_score = -0.001711705549079623, explained_variance_score = -8.50830329042207e-05
Evaluating 37 clusters: score = -0.0015118949844956653,
r2_score = -0.0015118949844956653, explained_variance_score = 0.0001119089841558063
Evaluating 38 clusters: score = -0.0011781147904363909,
r2_score = -0.0011781147904363909, explained_variance_score = 0.0004983032691938671
Evaluating 39 clusters: score = -0.00020629293582219432,
r2_score = -0.00020629293582219432, explained_variance_score = 0.0014503803243157698
```

In [96]:

```python
# display the resutls
plt.plot(range(2, num_clusters), scores)
plt.scatter(range(2, num_clusters), scores)
plt.grid()
_ =plt.xticks(range(2, num_clusters))
```



In [97]:

```python
# After learning the optimal value of k, we re-run the model
# This run of the model is using the optimal value of k
model = KNeighborsRegressor(n_neighbors=num_clusters, n_jobs=-1)
```

```
model.fit(X_train, y_train)

# gather the predictations that our model made for our test set
preds = model.predict(X_test)

# display the actuals and predictions for the test set
print('Actuals for test data set')
print(y_test)
print('Predictions for test data set')
print(preds)
```

```
Actuals for test data set
[0. 0. 0. ... 1. 0. 1.]
Predictions for test data set
[0.35  0.225 0.35  ... 0.125 0.175 0.225]
```

In [98]:

```
differs = y_test - preds

print(f'Differences between the two sets:\n{differs}\n')

print(f'r2_score: {r2_score(y_test,preds)}')
print(f'explained_variance_score = {explained_variance_score(y_test,preds)}')
```

```
Differences between the two sets:
[-0.35  -0.225 -0.35  ...  0.875 -0.175  0.775]

r2_score: -6.289089046029872e-05
explained_variance_score = 0.001561835242878784
```

So, for test_size = 0.15, score = +0.002

Now let's try test_size = 0.10

In [99]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10, random_state=42)
```

In [100…

```
# after creating the train/test data, I will iterate on the value of the clusters
# This will tell me what is the ideal value of k
# I will set the range of k values to test as 0-39, inclusive.
scores = []
num_clusters = 40
print(f'Features: {feat_cols} \nTarget: {target_col}')

# remember the ending number for range is not inclusive
for k in range(2, num_clusters):
    # output to let us know where we are
    # n_jobs=-1 will use all processors on your system
    model = KNeighborsRegressor(n_neighbors=k, n_jobs=-1)
    model.fit(X_train, y_train)
```

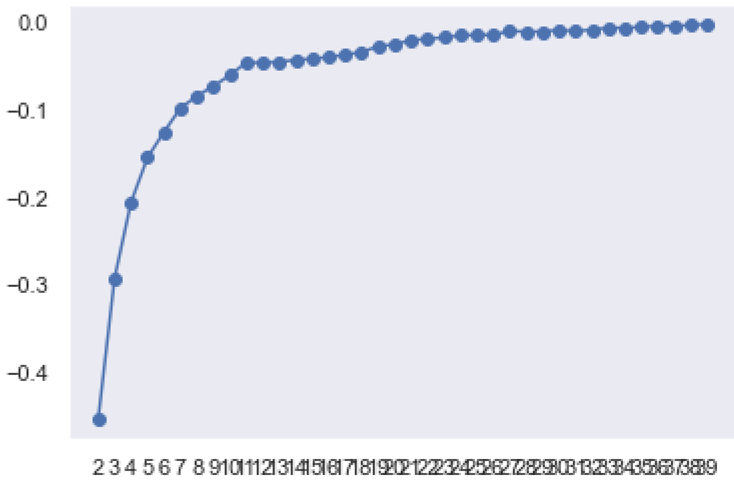
```
    scores.append(model.score(X_test, y_test))

    preds = model.predict(X_test)
    differs = y_test - preds

    print(f'Evaluating {k} clusters: score = {model.score(X_test, y_test)}, \nr2_score = {r2_score(y_test,preds)}, explained_varian
```

```
Features: ['age', 'marital', 'has_credit_in_default', 'has_housing_loan', 'has_personal_loan', 'contact_type', 'last_contact_month',
'last_contact_day_of_week', 'last_contact_duration', 'num_contacts_this_campaign', 'num_contacts_prev_campaign', 'emp.var.rate', 'co
ns.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed', 'job_admin.', 'job_blue-collar', 'job_entrepreneur', 'job_housemaid', 'j
ob_management', 'job_retired', 'job_self-employed', 'job_services', 'job_student', 'job_technician', 'job_unemployed', 'education_ba
sic.4y', 'education_basic.6y', 'education_basic.9y', 'education_high.school', 'education_illiterate', 'education_professional.cours
e', 'education_university.degree']
Target: subbed_term_deposit
Evaluating 2 clusters: score = -0.45477933890724276,
r2_score = -0.45477933890724276, explained_variance_score = -0.4456358195177139
Evaluating 3 clusters: score = -0.2924290098603959,
r2_score = -0.2924290098603959, explained_variance_score = -0.28446398852551735
Evaluating 4 clusters: score = -0.20561053373569838,
r2_score = -0.20561053373569838, explained_variance_score = -0.19686887766748673
Evaluating 5 clusters: score = -0.15451798225479974,
r2_score = -0.15451798225479974, explained_variance_score = -0.146753962818863
Evaluating 6 clusters: score = -0.12512141885791994,
r2_score = -0.12512141885791994, explained_variance_score = -0.11793699012789793
Evaluating 7 clusters: score = -0.09801907051304193,
r2_score = -0.09801907051304193, explained_variance_score = -0.09140053550035843
Evaluating 8 clusters: score = -0.08393907581470139,
r2_score = -0.08393907581470139, explained_variance_score = -0.07799201760723262
Evaluating 9 clusters: score = -0.07273574985855569,
r2_score = -0.07273574985855569, explained_variance_score = -0.06737868713437511
Evaluating 10 clusters: score = -0.060315841492262656,
r2_score = -0.060315841492262656, explained_variance_score = -0.05459330779670024
Evaluating 11 clusters: score = -0.0455111279063809,
r2_score = -0.0455111279063809, explained_variance_score = -0.040347096533253524
Evaluating 12 clusters: score = -0.04626997087619755,
r2_score = -0.04626997087619755, explained_variance_score = -0.0414516246365797
Evaluating 13 clusters: score = -0.04481209901038996,
r2_score = -0.04481209901038996, explained_variance_score = -0.039879622981913165
Evaluating 14 clusters: score = -0.042916502397649436,
r2_score = -0.042916502397649436, explained_variance_score = -0.038154182963663574
Evaluating 15 clusters: score = -0.04154906408881032,
r2_score = -0.04154906408881032, explained_variance_score = -0.03701484619906381
Evaluating 16 clusters: score = -0.03951044543614213,
r2_score = -0.03951044543614213, explained_variance_score = -0.035171224629555686
Evaluating 17 clusters: score = -0.03694317227248822,
r2_score = -0.03694317227248822, explained_variance_score = -0.03260395146590156
Evaluating 18 clusters: score = -0.03385960327200177,
r2_score = -0.03385960327200177, explained_variance_score = -0.029770693063873477
Evaluating 19 clusters: score = -0.02707572281068793,
```

```
r2_score = -0.02707572281068793, explained_variance_score = -0.023405897582340573
Evaluating 20 clusters: score = -0.02512946169881336,
r2_score = -0.02512946169881336, explained_variance_score = -0.02145760367045213
Evaluating 21 clusters: score = -0.019938581547195966,
r2_score = -0.019938581547195966, explained_variance_score = -0.01628529889691599
Evaluating 22 clusters: score = -0.018903887637156114,
r2_score = -0.018903887637156114, explained_variance_score = -0.015417311493705288
Evaluating 23 clusters: score = -0.01685484452128616,
r2_score = -0.01685484452128616, explained_variance_score = -0.013661378362199494
Evaluating 24 clusters: score = -0.014783611736148172,
r2_score = -0.014783611736148172, explained_variance_score = -0.01158608356627644
Evaluating 25 clusters: score = -0.014045891486869078,
r2_score = -0.014045891486869078, explained_variance_score = -0.010984137285741769
Evaluating 26 clusters: score = -0.014337037809051045,
r2_score = -0.014337037809051045, explained_variance_score = -0.011406853816661622
Evaluating 27 clusters: score = -0.009414815619327221,
r2_score = -0.009414815619327221, explained_variance_score = -0.006528271303056776
Evaluating 28 clusters: score = -0.01055248297195166,
r2_score = -0.01055248297195166, explained_variance_score = -0.007762569209922132
Evaluating 29 clusters: score = -0.010272218894991525,
r2_score = -0.010272218894991525, explained_variance_score = -0.007565712458349694
Evaluating 30 clusters: score = -0.008547863069909178,
r2_score = -0.008547863069909178, explained_variance_score = -0.005854724374663167
Evaluating 31 clusters: score = -0.009854393047142862,
r2_score = -0.009854393047142862, explained_variance_score = -0.007092682633430103
Evaluating 32 clusters: score = -0.008073361535258394,
r2_score = -0.008073361535258394, explained_variance_score = -0.005314427143302325
Evaluating 33 clusters: score = -0.005930580928306339,
r2_score = -0.005930580928306339, explained_variance_score = -0.003181010125979933
Evaluating 34 clusters: score = -0.0062025733720332266,
r2_score = -0.0062025733720332266, explained_variance_score = -0.003490084054792675
Evaluating 35 clusters: score = -0.0049329640601660785,
r2_score = -0.0049329640601660785, explained_variance_score = -0.002286531343296261
Evaluating 36 clusters: score = -0.003902102078833458,
r2_score = -0.003902102078833458, explained_variance_score = -0.0011876835373882955
Evaluating 37 clusters: score = -0.003634876607422566,
r2_score = -0.003634876607422566, explained_variance_score = -0.0009255533803100846
Evaluating 38 clusters: score = -0.0031282927886873146,
r2_score = -0.0031282927886873146, explained_variance_score = -0.0004816408919334769
Evaluating 39 clusters: score = -0.0015984355129907701,
r2_score = -0.0015984355129907701, explained_variance_score = 0.0011355736697032937
```

In [101…

```
# display the resutls
plt.plot(range(2, num_clusters), scores)
plt.scatter(range(2, num_clusters), scores)
plt.grid()
_ =plt.xticks(range(2, num_clusters))
```



In [102…

```
# After learning the optimal value of k, we re-run the model
# This run of the model is using the optimal value of k
model = KNeighborsRegressor(n_neighbors=num_clusters, n_jobs=-1)
model.fit(X_train, y_train)

# gather the predictations that our model made for our test set
preds = model.predict(X_test)

# display the actuals and predictions for the test set
print('Actuals for test data set')
print(y_test)
print('Predictions for test data set')
print(preds)
```

```
Actuals for test data set
[0. 0. 0. ... 1. 0. 0.]
Predictions for test data set
[0.3   0.25  0.35  ... 0.325 0.275 0.475]
```

In [103…

```
differs = y_test - preds

print(f'Differences between the two sets:\n{differs}\n')

print(f'r2_score: {r2_score(y_test,preds)}')
print(f'explained_variance_score = {explained_variance_score(y_test,preds)}')
```

```
Differences between the two sets:
[-0.3   -0.25  -0.35  ...  0.675 -0.275 -0.475]

r2_score: -0.0018029991747001706
explained_variance_score = 0.000961053185431493
```

So, for test_size = 0.1, score = +0.001

- So, from the four overall values of test_size tested, test_size=0.15 provided the highest overall model score. The score achieved was positive which is a good sign. However, it's still a low value, only +0.002. However, it's a positive change!

K-fold Cross-Validation Analysis

- Complete a K-fold cross-validation analysis for your improved model
  - You need to use at less three additional models
  - Discuss the difference in the performance of the 4 algorithms against your improved dataset.

In [104…

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error

from matplotlib import pyplot
```

In [105…

```
# Spot Check Algorithms
models = []
models.append(('LR', LinearRegression()))
models.append(('LASSO', Lasso()))
models.append(('EN', ElasticNet()))
models.append(('KNN', KNeighborsRegressor()))
models.append(('CART', DecisionTreeRegressor()))
#models.append(('SVR', SVR(gamma='auto')))
```

In [106…

```
# evaluate each model in turn
seed = 42
num_folds = 5
scoring = 'neg_mean_squared_error'
```

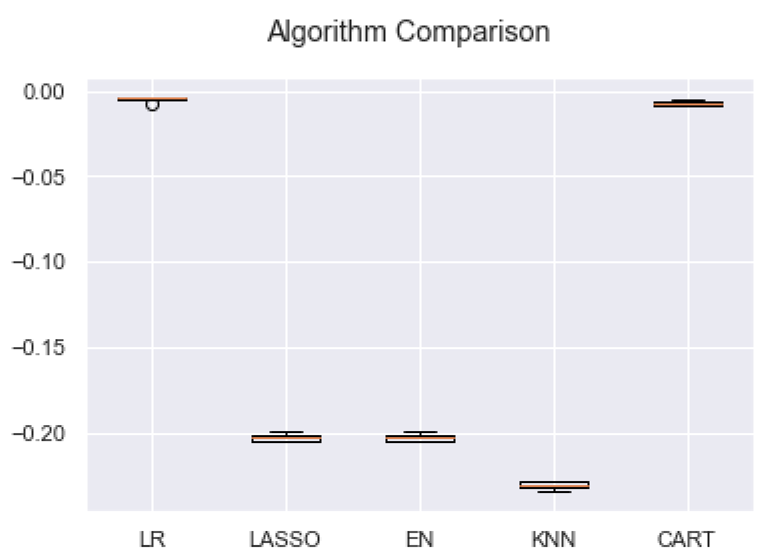
In [107…

```
results = []
names = []
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=42)
for name, model in models:
    kfold = KFold(n_splits=num_folds, random_state=seed, shuffle=True)
    cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: -0.005188 (0.001513)
LASSO: -0.202414 (0.002086)
EN: -0.202425 (0.002082)
KNN: -0.230479 (0.002370)
CART: -0.007038 (0.001468)
```

In [108…

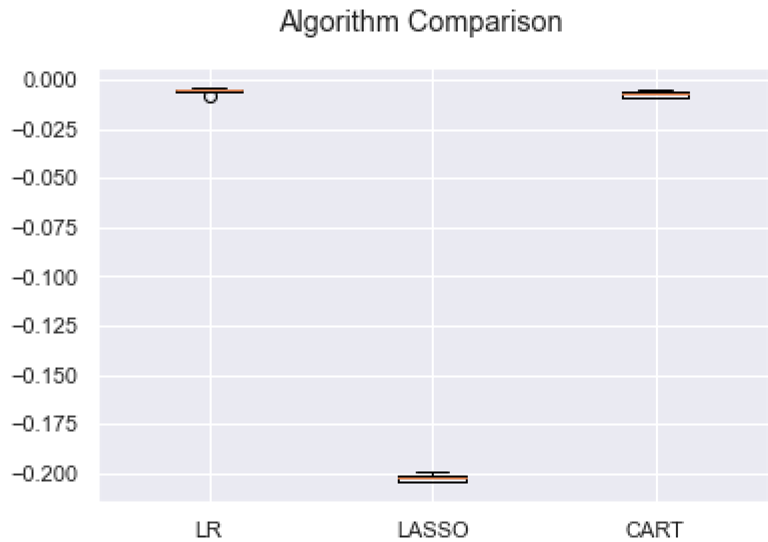
```
# Compare Algorithms
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```



In [109…

```
#Now I will drop the 2 lowest-scoring algorithms
#I will drop the EN and KNN algos
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
```

```
ax = fig.add_subplot(111)
pyplot.boxplot([results[0], results[1], results[4]])
ax.set_xticklabels([names[0], names[1], names[4]])
pyplot.show()
```



It is strange, all algorithms sampled still have average scores which were negative. This is disappointing as it points to an inaccurate model. I'm not sure what else can be changed with this specific algorithm comparison to improve the scores. The Linear Regresson algorithm attained the highest score with an average score of -0.005. This was followed by the Decision Tree Regressor with an average score of -0.007. Then scores dropped semi-sharply. All remaining 3 algorithms had scores in the range of ~-0.20. The Lasso algorithm was 3rd highest with an average score of -0.20, the Elastic Net algorithm was 4th with an average score of -0.20, and the KNN algorithm was last with an average score of -0.23.

In [ ]:

```

```

V. Conclusion

In conclusion, the Linear Regression algorithm achieved the highest score when compared against the Lasso, Elastic Net, KNN< and Decision Tree Regressor algorithms. However, this must be taken with a grain of salt as all average scores were negative, which indicates poor model performance. Future steps to be taken with this model can be steps to improve the model performance, such as further experimenting with the test_size proportion, the value of k, and including more data with the process, possibly from a more detailed data cleaning.

Thank you!

VI. References

1) Dolon, B. (2022, January 6). An Easy Way to Replace Values in a Pandas DataFrame. Medium. Retrieved April 10, 2022, from https://towardsdatascience.com/an-easy-way-to-replace-values-in-a-pandas-dataframe-2826bd34e59a

2) GeeksforGeeks. (2018, February 6). Replacing strings with numbers in Python for Data Analysis. Retrieved April 10, 2022, from https://www.geeksforgeeks.org/replacing-strings-with-numbers-in-python-for-data-analysis/

3) Class dataset provided for this assignment: bank-additional-full.csv, bank-additional-names.txt

4) From the Experts PDF: Week 5

5) Week 5 Assignment Lab (Jupyter notebook)

In [ ]:

```

```