In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Project Summary: You are contracted to assist a bank with a new marketing campaign for clients who will have a higher probability in signing up for a Term Deposit.

- The data is provided below and the description of the features are also added to the assist you with your model
- Provide analysis and visualization of the data set
- What can you determine from your initial analysis and what feed back can oyu five the bank in the clients privided
- How accurate is your model and does it provice resonable accuracy in this scenario
- Save your model and the history for the back to inplimant in their marketing campaign

Target Class Definition

- A term deposit refers to when you lock your money in an account for a certain period of time and at a specified interest rate. You will not be able to access your money for the length of the agreed term without incurring a penalty fee.
- The target class is to determine if a client will subscribe to a Term Deposite based on certain features
- y - has the client subscribed a term deposit? (binary: "yes","no")
- All features details are shown below

Features Details

- Each column will be describe below

In [2]:
```
text = open("resources/bank-additional-names.txt", mode="r").read()
```

In [3]:
```
print(text[0:])
```

```
Citation Request:
  This dataset is publicly available for research. The details are desc
ribed in [Moro et al., 2014].
  Please include this citation if you plan to use this database:

  [Moro et al., 2014] S. Moro, P. Cortez and P. Rita. A Data-Driven App
roach to Predict the Success of Bank Telemarketing. Decision Support Sy
stems, In press, http://dx.doi.org/10.1016/j.dss.2014.03.001

  Available at: [pdf] http://dx.doi.org/10.1016/j.dss.2014.03.001
                [bib] http://www3.dsi.uminho.pt/pcortez/bib/2014-dss.tx
t

1. Title: Bank Marketing (with social/economic context)

2. Sources
   Created by: Sérgio Moro (ISCTE-IUL), Paulo Cortez (Univ. Minho) and
Paulo Rita (ISCTE-IUL) @ 2014

3. Past Usage:

  The full dataset (bank-additional-full.csv) was described and analyze
d in:

  S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the
Success of Bank Telemarketing. Decision Support Systems (2014), doi:10.
1016/j.dss.2014.03.001.

4. Relevant Information:

   This dataset is based on "Bank Marketing" UCI dataset (please check
the description at: http://archive.ics.uci.edu/ml/datasets/Bank+Marketi
ng).
   The data is enriched by the addition of five new social and economic
features/attributes (national wide indicators from a ~10M population co
untry), published by the Banco de Portugal and publicly available at: h
ttps://www.bportugal.pt/estatisticasweb.
   This dataset is almost identical to the one used in [Moro et al., 20
14] (it does not include all attributes due to privacy concerns).
   Using the rminer package and R tool (http://cran.r-project.org/web/p
ackages/rminer/), we found that the addition of the five new social and
economic attributes (made available here) lead to substantial improveme
nt in the prediction of a success, even when the duration of the call i
s not included. Note: the file can be read in R using: d=read.table("ba
nk-additional-full.csv",header=TRUE,sep=";")

   The zip file includes two datasets:
      1) bank-additional-full.csv with all examples, ordered by date (f
rom May 2008 to November 2010).
      2) bank-additional.csv with 10% of the examples (4119), randomly
selected from bank-additional-full.csv.
   The smallest dataset is provided to test more computationally demand
ing machine learning algorithms (e.g., SVM).

   The binary classification goal is to predict if the client will subs
cribe a bank term deposit (variable y).
```

```
5. Number of Instances: 41188 for bank-additional-full.csv

6. Number of Attributes: 20 + output attribute.

7. Attribute information:

   For more information, read [Moro et al., 2014].

   Input variables:
   # bank client data:
   1 - age (numeric)
   2 - job : type of job (categorical: "admin.","blue-collar","entrepre
neur","housemaid","management","retired","self-employed","services","st
udent","technician","unemployed","unknown")
   3 - marital : marital status (categorical: "divorced","married","sin
gle","unknown"; note: "divorced" means divorced or widowed)
   4 - education (categorical: "basic.4y","basic.6y","basic.9y","high.s
chool","illiterate","professional.course","university.degree","unknow
n")
   5 - default: has credit in default? (categorical: "no","yes","unknow
n")
   6 - housing: has housing loan? (categorical: "no","yes","unknown")
   7 - loan: has personal loan? (categorical: "no","yes","unknown")
   # related with the last contact of the current campaign:
   8 - contact: contact communication type (categorical: "cellular","te
lephone")
   9 - month: last contact month of year (categorical: "jan", "feb", "m
ar", ..., "nov", "dec")
  10 - day_of_week: last contact day of the week (categorical: "mon","t
ue","wed","thu","fri")
  11 - duration: last contact duration, in seconds (numeric). Important
note:  this attribute highly affects the output target (e.g., if durati
on=0 then y="no"). Yet, the duration is not known before a call is perf
ormed. Also, after the end of the call y is obviously known. Thus, this
input should only be included for benchmark purposes and should be disc
arded if the intention is to have a realistic predictive model.
   # other attributes:
  12 - campaign: number of contacts performed during this campaign and
for this client (numeric, includes last contact)
  13 - pdays: number of days that passed by after the client was last c
ontacted from a previous campaign (numeric; 999 means client was not pr
eviously contacted)
  14 - previous: number of contacts performed before this campaign and
for this client (numeric)
  15 - poutcome: outcome of the previous marketing campaign (categorica
l: "failure","nonexistent","success")
   # social and economic context attributes
  16 - emp.var.rate: employment variation rate - quarterly indicator (n
umeric)
  17 - cons.price.idx: consumer price index - monthly indicator (numeri
c)
  18 - cons.conf.idx: consumer confidence index - monthly indicator (nu
meric)
  19 - euribor3m: euribor 3 month rate - daily indicator (numeric)
  20 - nr.employed: number of employees - quarterly indicator (numeric)

  Output variable (desired target):
```

```
21 - y - has the client subscribed a term deposit? (binary: "yes","n
o")

8. Missing Attribute Values: There are several missing values in some c
ategorical attributes, all coded with the "unknown" label. These missin
g values can be treated as a possible class label or using deletion or
imputation techniques.
```

In [4]:
```
df = pd.read_csv("resources/bank.csv")
```

In [5]:
```
df
```

Out[5]:

	age	job	marital	education	default	balance	housing	loan	contact	day	mon
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	c
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	m
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	a
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	j
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	m
...	...	...	...	...	...	...	...	...	...	...	
4516	33	services	married	secondary	no	-333	yes	no	cellular	30	
4517	57	self-employed	married	tertiary	yes	-3313	yes	yes	unknown	9	m
4518	57	technician	married	secondary	no	295	no	no	cellular	19	a
4519	28	blue-collar	married	secondary	no	1137	no	no	cellular	6	f
4520	44	entrepreneur	single	tertiary	no	1136	yes	yes	cellular	3	a

4521 rows × 17 columns

DATA EVALUATION

- Checking for missing data, null values
- Describe - Min max values
- Data Frame length and coulmn Count etc

info

- Has 4521 clients and 17 columns
- 16 comumnd of features and 1 for our target y-> Term Deposit Account

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4521 entries, 0 to 4520
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        4521 non-null   int64 
 1   job        4521 non-null   object
 2   marital    4521 non-null   object
 3   education  4521 non-null   object
 4   default    4521 non-null   object
 5   balance    4521 non-null   int64 
 6   housing    4521 non-null   object
 7   loan       4521 non-null   object
 8   contact    4521 non-null   object
 9   day        4521 non-null   int64 
 10  month      4521 non-null   object
 11  duration   4521 non-null   int64 
 12  campaign   4521 non-null   int64 
 13  pdays      4521 non-null   int64 
 14  previous   4521 non-null   int64 
 15  poutcome   4521 non-null   object
 16  y          4521 non-null   object
dtypes: int64(7), object(10)
memory usage: 600.6+ KB
```

In [7]:
```
df.describe()
```

Out[7]:

	age	balance	day	duration	campaign	pdays	previ
count	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000
mean	41.170095	1422.657819	15.915284	263.961292	2.793630	39.766645	0.542
std	10.576211	3009.638142	8.247667	259.856633	3.109807	100.121124	1.693
min	19.000000	-3313.000000	1.000000	4.000000	1.000000	-1.000000	0.000
25%	33.000000	69.000000	9.000000	104.000000	1.000000	-1.000000	0.000
50%	39.000000	444.000000	16.000000	185.000000	2.000000	-1.000000	0.000
75%	49.000000	1480.000000	21.000000	329.000000	3.000000	-1.000000	0.000
max	87.000000	71188.000000	31.000000	3025.000000	50.000000	871.000000	25.000

Converting Columns

- Since we need numeric values for out model we will need to convert the values to one hot encoding
- This will assit with processing the data and our Visualization

Jobs

- Will one hot encode then drop the jobs columns
- Will drop the first column because it will be a perfect predictor of the remaining columns

In [8]:
```
df["job"].nunique()
```

Out[8]:
```
12
```

In [9]:
```
df["job"].unique()
```

Out[9]:
```
array(['unemployed', 'services', 'management', 'blue-collar',
       'self-employed', 'technician', 'entrepreneur', 'admin.', 'studen
t',
       'housemaid', 'retired', 'unknown'], dtype=object)
```

In [10]:
```
jobs_oneHot = pd.get_dummies(df["job"], drop_first=True)
```

In [11]:
```
df.drop("job", axis=1, inplace=True)
```

In [12]:
```
df = pd.concat([df,jobs_oneHot], axis=1)
```

Marital Status

- unique values for this feature

In [13]:
```
df["marital"].nunique()
```

Out[13]:
```
3
```

In [14]:
```
df["marital"].unique()
```

Out[14]:
```
array(['married', 'single', 'divorced'], dtype=object)
```

In [15]:
```
marital = pd.get_dummies(df["marital"], drop_first=True)
```

In [16]:
```
df.drop("marital", axis=1, inplace=True)
```

In [17]:
```
df = pd.concat([df, marital], axis=1)
```

In [18]:
```
df.head()
```

Out[18]:

	age	education	default	balance	housing	loan	contact	day	month	duration	...	manage
0	30	primary	no	1787	no	no	cellular	19	oct	79	...	
1	33	secondary	no	4789	yes	yes	cellular	11	may	220	...	
2	35	tertiary	no	1350	yes	no	cellular	16	apr	185	...	
3	30	tertiary	no	1476	yes	yes	unknown	3	jun	199	...	
4	59	secondary	no	0	yes	no	unknown	5	may	226	...	

5 rows × 28 columns

Education

- 4 uniqie value for this feature

In [19]:
```
df["education"].nunique()
```

Out[19]: 4

In [20]:
```
df["education"].unique()
```

Out[20]: array(['primary', 'secondary', 'tertiary', 'unknown'], dtype=object)

In [21]:
```
ed = pd.get_dummies(df["education"], drop_first= True)
```

In [22]:
```
df = pd.concat([df, ed], axis=1)
```

In [23]:
```
df.drop("education", axis=1, inplace=True)
```

In [24]:
```
df.head()
```

Out[24]:

	age	default	balance	housing	loan	contact	day	month	duration	campaign	...	services
0	30	no	1787	no	no	cellular	19	oct	79	1	...	0
1	33	no	4789	yes	yes	cellular	11	may	220	1	...	1
2	35	no	1350	yes	no	cellular	16	apr	185	1	...	0
3	30	no	1476	yes	yes	unknown	3	jun	199	4	...	0
4	59	no	0	yes	no	unknown	5	may	226	1	...	0

5 rows × 30 columns

Default

- If a client has defaulted
- since this is binary we can simply convert and set the new comumn without concatination

In [25]:
```
default = pd.get_dummies(df["default"], drop_first=True)
```

In [26]:
```
df["default"] = default
```

Housing

- We will use binary concatination to the column

In [27]:
```
housing = pd.get_dummies(df["housing"], drop_first=True)
```

In [28]:
```
df["housing"] = housing
```

In [29]:
```
df.head()
```

Out[29]:

	age	default	balance	housing	loan	contact	day	month	duration	campaign	...	services
0	30	0	1787	0	no	cellular	19	oct	79	1	...	0
1	33	0	4789	1	yes	cellular	11	may	220	1	...	1
2	35	0	1350	1	no	cellular	16	apr	185	1	...	0
3	30	0	1476	1	yes	unknown	3	jun	199	4	...	0
4	59	0	0	1	no	unknown	5	may	226	1	...	0

5 rows × 30 columns

Loan

- If the client has a personal loan
- Taking the binary concatination approach here as well

In [30]:
```
loan = pd.get_dummies(df["loan"], drop_first=True)
```

In [31]:
```
df["loan"] = loan
```

In [32]:
```
df.head()
```

Out[32]:

	age	default	balance	housing	loan	contact	day	month	duration	campaign	...	services
0	30	0	1787	0	0	cellular	19	oct	79	1	...	0
1	33	0	4789	1	1	cellular	11	may	220	1	...	1
2	35	0	1350	1	0	cellular	16	apr	185	1	...	0
3	30	0	1476	1	1	unknown	3	jun	199	4	...	0
4	59	0	0	1	0	unknown	5	may	226	1	...	0

5 rows × 30 columns

contact type

- How the client was contacted or means of communication

In [33]:
```
df["contact"].nunique()
```

Out[33]: 3

In [34]:
```
contact = pd.get_dummies(df["contact"], drop_first=True)
```

In [35]:
```
df= pd.concat([df, contact], axis=1)
```

In [36]:
```
df.drop("contact", axis=1, inplace=True)
```

In [37]:
```
df.head()
```

Out[37]:

	age	default	balance	housing	loan	day	month	duration	campaign	pdays	...	technician
0	30	0	1787	0	0	19	oct	79	1	-1	...	0
1	33	0	4789	1	1	11	may	220	1	339	...	0
2	35	0	1350	1	0	16	apr	185	1	330	...	0
3	30	0	1476	1	1	3	jun	199	4	-1	...	0
4	59	0	0	1	0	5	may	226	1	-1	...	0

5 rows × 31 columns

In [38]:
```
df["month"].unique()
```

Out[38]:
```
array(['oct', 'may', 'apr', 'jun', 'feb', 'aug', 'jan', 'jul', 'nov',
       'sep', 'mar', 'dec'], dtype=object)
```

Converting Month

- last contact month
- we can create a dictionary to replace the values for thre months

In [39]:
```
months = [10,5,4,6,2,8,1,7,11,9,3,12]
```

In [40]:
```
mm_t = list(df["month"].unique())
```

In [41]:
```
mm_t
```

Out[41]:
```
['oct',
 'may',
 'apr',
 'jun',
 'feb',
 'aug',
 'jan',
 'jul',
 'nov',
 'sep',
 'mar',
 'dec']
```

In [42]:
```
num_mon = dict(zip(mm_t, months))
```

In [43]:
```
num_mon
```

Out[43]:
```
{'oct': 10,
 'may': 5,
 'apr': 4,
 'jun': 6,
 'feb': 2,
 'aug': 8,
 'jan': 1,
 'jul': 7,
 'nov': 11,
 'sep': 9,
 'mar': 3,
 'dec': 12}
```

In [44]:
```
df["month"] = df["month"].apply(lambda x: num_mon[x])
```

Setting Target To one hot coding

- Appears the target feature is unbalanced
- This will cause the model to better perform at prediction for one class vs the other

In [45]:
```
target = pd.get_dummies(df["y"], drop_first=True)
```

In [46]:
```
df["y"].nunique()
```

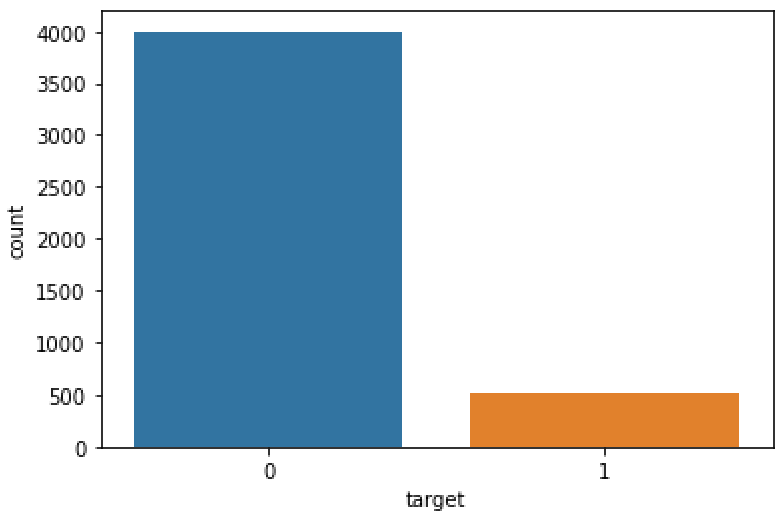
Out[46]:
```
2
```

In [47]:
```
df["y"].value_counts()
```

Out[47]:
```
no     4000
yes     521
Name: y, dtype: int64
```

In [52]:
```
sns.countplot(df["target"])
```

Out[52]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fde15b484f0>
```



In [53]:
```
df["target"] = target
```

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

Out[56]:

	age	default	balance	housing	loan	day	month	duration	campaign	pdays	...	unemploye
0	30	0	1787	0	0	19	10	79	1	-1	...	
1	33	0	4789	1	1	11	5	220	1	339	...	
2	35	0	1350	1	0	16	4	185	1	330	...	
3	30	0	1476	1	1	3	6	199	4	-1	...	
4	59	0	0	1	0	5	5	226	1	-1	...	

5 rows × 31 columns

Checking correlation

- This will tell us if there is a future that is a perfect predictor of the target
- also if there was a mistake somewhere in eliminating features that was converted

In [57]:
```
df.corrwith(df["target"]).sort_values(ascending =False)
```

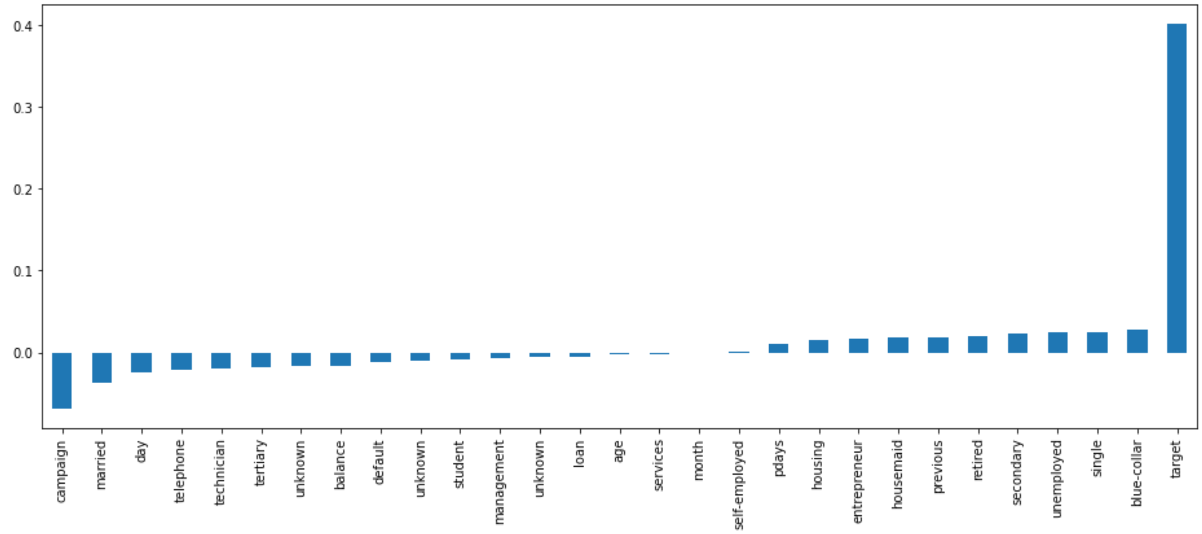
Out[57]:
```
target           1.000000
duration         0.401118
previous         0.116714
pdays            0.104087
retired          0.086675
tertiary         0.056649
student          0.047809
single           0.045815
age              0.045092
management       0.032634
telephone        0.025878
month            0.023335
unknown          0.019886
balance          0.017905
housemaid        0.004872
default          0.001303
self-employed   -0.003827
unemployed      -0.007312
unknown         -0.008870
technician      -0.010154
day             -0.011244
entrepreneur    -0.015968
services        -0.024071
secondary       -0.028744
campaign        -0.061147
married         -0.064643
blue-collar     -0.068147
loan            -0.070517
housing         -0.104683
unknown         -0.139399
dtype: float64
```

Duration

- Is the highest correlation to the Target
- This is the strongest predictor a client will have a term Deposit
- but is this a feasible feature to add. Lets think of why this could be an issue

In [58]:
```
df.corrwith(df["duration"]).sort_values(ascending =True)[:-1].plot(kind
="bar", figsize = (16,6))
```

Out[58]: <matplotlib.axes._subplots.AxesSubplot at 0x7fde15fe2d30>



Looking at Campaign vs Duration

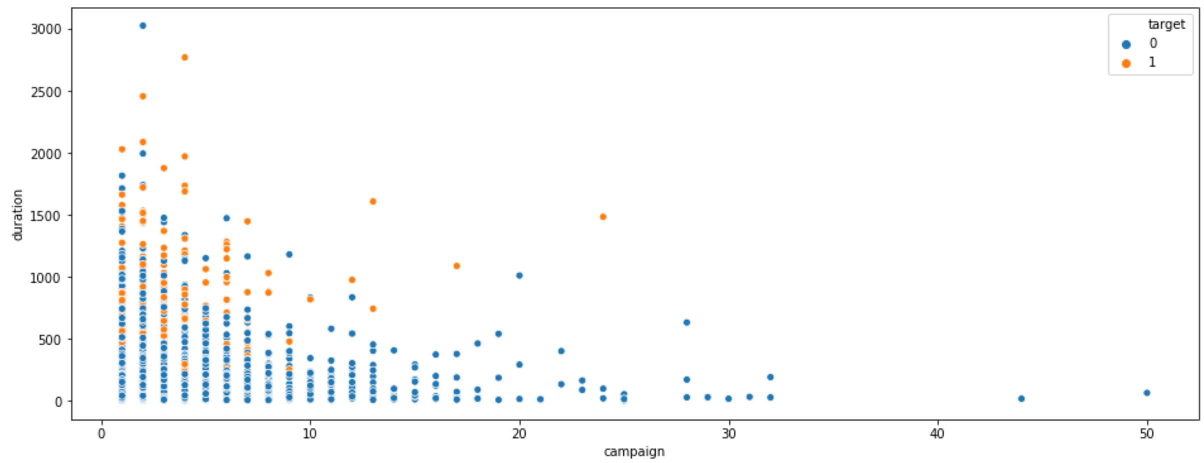
- Campaign - number of contacts performed during this campaign and for this client
- Duration - last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y="no"). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.
  - We will remove this value for our model

Appears the as the campaign increases

- There is also a slim change the user will subscribe to a term deposit
- The higher the duration the the more likely a term deposit subscription will occurr
- We will consider the suggesiton to remove this feature for a more realistic model

In [59]:
```
plt.figure(figsize=(16,6))
sns.scatterplot(x =df["campaign"], y= df['duration'], hue=df["target"])
```

Out[59]: <matplotlib.axes._subplots.AxesSubplot at 0x7fde15e4d1c0>



In [60]:
```
df.drop("duration", axis=1, inplace=True)
```

In [61]:
```
df.head()
```

Out[61]:

	age	default	balance	housing	loan	day	month	campaign	pdays	previous	...	unemploye
0	30	0	1787	0	0	19	10	1	-1	0	...	
1	33	0	4789	1	1	11	5	1	339	4	...	
2	35	0	1350	1	0	16	4	1	330	1	...	
3	30	0	1476	1	1	3	6	4	-1	0	...	
4	59	0	0	1	0	5	5	1	-1	0	...	

5 rows × 30 columns

Preparing the data

- now that we have a data frame of the features we will use for the model
- Lets prepare this data for training and testing
- We will begin with SKLearn logistic regression for classification

Checking df

- Making sure all values are numeric
- looks like we missed on feature
-
  - lets convert this below

In [62]:
```
pout = pd.get_dummies(df["poutcome"], drop_first=True)
```

In [63]:
```
df = pd.concat([df, pout], axis=1)
```

In [64]:
```
df.drop("poutcome", axis=1, inplace=True)
```

In [65]:
```
df.head()
```

Out[65]:

	age	default	balance	housing	loan	day	month	campaign	pdays	previous	...	single	se
0	30	0	1787	0	0	19	10	1	-1	0	...	0	
1	33	0	4789	1	1	11	5	1	339	4	...	0	
2	35	0	1350	1	0	16	4	1	330	1	...	1	
3	30	0	1476	1	1	3	6	4	-1	0	...	0	
4	59	0	0	1	0	5	5	1	-1	0	...	0	

5 rows × 32 columns

In [66]:
```
df.select_dtypes(exclude="int")
```

Out[66]:

	default	housing	loan	blue-collar	entrepreneur	housemaid	management	retired	self-employed
0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0
2	0	1	0	0	0	0	1	0	0
3	0	1	1	0	0	0	1	0	0
4	0	1	0	1	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...
4516	0	1	0	0	0	0	0	0	0
4517	1	1	1	0	0	0	0	0	1
4518	0	0	0	0	0	0	0	0	0
4519	0	0	0	1	0	0	0	0	0
4520	0	1	1	0	1	0	0	0	0

4521 rows × 25 columns

In [67]:
```
df["target"].value_counts()
```

Out[67]:
```
0    4000
1     521
Name: target, dtype: int64
```

In [83]:
```
X = df.drop("target", axis=1)
y = df["target"]
```

Training testing and Splitting

- 80% training and 20% testing

In [84]:
```
from sklearn.model_selection import train_test_split
```

In [85]:
```
 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
0, random_state=42)
```

Scaling Data | Normalizing

In [86]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [87]:
```
scalar = MinMaxScaler()
```

In [88]:
```
X_train = scalar.fit_transform(X_train)
```

In [89]:
```
X_test = scalar.transform(X_test)
```

In [90]:
```
X_train.shape
```

Out[90]:
```
(3616, 31)
```

In [91]:
```
X_test.shape
```

Out[91]:
```
(905, 31)
```

Importing Model and libraries

In [92]:
```
from sklearn.linear_model import LogisticRegression
```

In [93]:
```
model = LogisticRegression()
```

In [94]:
```
model.fit(X_train,y_train)
```

Out[94]:
```
LogisticRegression()
```

In [95]:
```
model.classes_
```

Out[95]:
```
array([0, 1], dtype=uint8)
```

Making Predicitons

- using the testing data since the model has not see this data previously

In [96]:
```
pred = model.predict(X_test)
```

Metrics

- A Classificaiton Report will be generated to see how the model predictions performed in its pedicitons

In [97]:
```
from sklearn.metrics import classification_report, confusion_matrix, exp
lained_variance_score
```

Model is 80+ % accurate

- This is good model for a client probability of subscribing to a Term Deposit
- Though the model did much better at classifying the No (0) vs Yes(1)
- Again keeping in mind that the data was a balanced favoring the NO

In [100]:
```
print(confusion_matrix(y_test, pred))
```

```
[[799   8]
 [ 85  13]]
```

In [101]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.99      0.95       807
           1       0.62      0.13      0.22        98

    accuracy                           0.90       905
   macro avg       0.76      0.56      0.58       905
weighted avg       0.87      0.90      0.87       905
```

Random Client

- We want to see what happens when a new client is provided to evaluate
- Lets take a look at how the model will predict
- Since we do not have new client data we will pass in the a random client from the data frame

In [138]:
```
from random import randint
random_index = randint(1, len(df))
random_client = df.drop("target", axis=1).iloc[random_index]
```

Collecting the values of the client

- also the values must be reshaped to the training shape we trained our model on

In [139]:
```
X_train.shape
```

Out[139]:
```
(3616, 31)
```

In [140]:
```
random_client = random_client.values.reshape(1,31)
```

In [141]:
```
random_client
```

Out[141]:
```
array([[ 34,   0, 209,   1,   1,   8,   4,   2,  -1,   0,   0,   0,
0,
          0,   0,   0,   0,   0,   1,   0,   0,   1,   0,   1,   0,
0,
          0,   0,   0,   0,   1]])
```

Making Prediction on random Client

In [142]:
```
model.predict(random_client)
```

Out[142]:
```
array([0], dtype=uint8)
```

Checking True Value

- Will grab form the true data frame

In [143]:
```
df.iloc[random_index]["target"]
```

Out[143]:
```
0
```

Deep AAN

- Applyinf a Deep Network for potential improvments in the prediciton of our model

In [237]:
```
X = df.drop("target", axis=1).values
y = df["target"].values
```

In [238]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20
, random_state=42)
```

Scaling the data

In [239]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [240]:
```
scalar = MinMaxScaler()
```

In [241]:
```
X_train = scalar.fit_transform(X_train)
```

In [242]:
```
X_test = scalar.transform(X_test)
```

Importing TensorFlow and Keras libraries

In [243]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Dense
from tensorflow.keras.callbacks import EarlyStopping
```

In [244]:
```
stop = EarlyStopping(monitor="val_loss", mode="min", patience=10)
```

In [245]:
```
X_train.shape
```

Out[245]:
```
(3616, 31)
```

Adding a Dropout and a Dense layer

- The Dropout layer will prevent over training to the noise of the features and turning off some neurons will slow the process or prevent this over all
- Early stopping will check if the model shows no improvement for a certain patience(epoch) -> if no stop training. At this point the model will also begin to overtrain

In [246]:
```
model = Sequential()
model.add(Dense(units = 30, activation ="relu"))
model.add(Dropout(0.25))
model.add(Dense(units =30, activation ="relu"))
model.add(Dropout(0.25))
model.add(Dense(units =30, activation ="relu"))
model.add(Dropout(0.25))
model.add(Dense(units = 20, activation ="relu"))
model.add(Dropout(0.25))
model.add(Dense(units = 10, activation ="relu"))
model.add(Dense(units = 1, activation ="sigmoid"))
model.compile(optimizer  = "adam", loss = "binary_crossentropy")
```

In [247]:
```
model.fit(X_train,y_train, validation_data=(X_test,y_test), epochs=22, c
allbacks=[stop])
```

```
Epoch 1/22
113/113 [==============================] - 1s 4ms/step - loss: 0.5969 -
val_loss: 0.3361
Epoch 2/22
113/113 [==============================] - 0s 3ms/step - loss: 0.3647 -
val_loss: 0.3382
Epoch 3/22
113/113 [==============================] - 0s 3ms/step - loss: 0.3849 -
val_loss: 0.3271
Epoch 4/22
113/113 [==============================] - 0s 2ms/step - loss: 0.3387 -
val_loss: 0.3257
Epoch 5/22
113/113 [==============================] - 0s 3ms/step - loss: 0.3260 -
val_loss: 0.3315
Epoch 6/22
113/113 [==============================] - 0s 3ms/step - loss: 0.3524 -
val_loss: 0.3262
Epoch 7/22
113/113 [==============================] - 0s 3ms/step - loss: 0.3324 -
val_loss: 0.3198
Epoch 8/22
113/113 [==============================] - 1s 6ms/step - loss: 0.3344 -
val_loss: 0.3301
Epoch 9/22
113/113 [==============================] - 0s 3ms/step - loss: 0.3350 -
val_loss: 0.3252
Epoch 10/22
113/113 [==============================] - 0s 3ms/step - loss: 0.3430 -
val_loss: 0.3195
Epoch 11/22
113/113 [==============================] - 0s 4ms/step - loss: 0.3302 -
val_loss: 0.3230
Epoch 12/22
113/113 [==============================] - 0s 3ms/step - loss: 0.3091 -
val_loss: 0.3301
Epoch 13/22
113/113 [==============================] - 0s 3ms/step - loss: 0.3153 -
val_loss: 0.3250
Epoch 14/22
113/113 [==============================] - 0s 3ms/step - loss: 0.3154 -
val_loss: 0.3201
Epoch 15/22
113/113 [==============================] - 0s 4ms/step - loss: 0.3147 -
val_loss: 0.3297
Epoch 16/22
113/113 [==============================] - 0s 4ms/step - loss: 0.3205 -
val_loss: 0.3182
Epoch 17/22
113/113 [==============================] - 1s 6ms/step - loss: 0.3358 -
val_loss: 0.3218
Epoch 18/22
113/113 [==============================] - 0s 3ms/step - loss: 0.3146 -
val_loss: 0.3369
Epoch 19/22
113/113 [==============================] - 0s 3ms/step - loss: 0.3093 -
val_loss: 0.3245
```

```
Epoch 20/22
113/113 [==============================] - 1s 6ms/step - loss: 0.3036 -
val_loss: 0.3286
Epoch 21/22
113/113 [==============================] - 1s 6ms/step - loss: 0.3216 -
val_loss: 0.3263
Epoch 22/22
113/113 [==============================] - 0s 3ms/step - loss: 0.3149 -
val_loss: 0.3205
```

Out[247]: <tensorflow.python.keras.callbacks.History at 0x7fde077565b0>

In [248]:
```
model.summary()
```

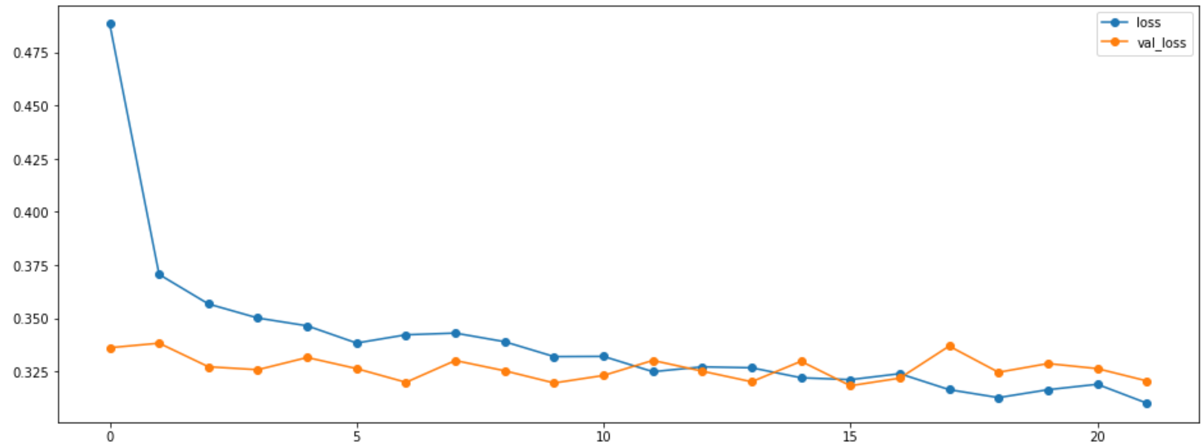
```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_42 (Dense)             (None, 30)                960
_________________________________________________________________
dropout_28 (Dropout)         (None, 30)                0
_________________________________________________________________
dense_43 (Dense)             (None, 30)                930
_________________________________________________________________
dropout_29 (Dropout)         (None, 30)                0
_________________________________________________________________
dense_44 (Dense)             (None, 30)                930
_________________________________________________________________
dropout_30 (Dropout)         (None, 30)                0
_________________________________________________________________
dense_45 (Dense)             (None, 20)                620
_________________________________________________________________
dropout_31 (Dropout)         (None, 20)                0
_________________________________________________________________
dense_46 (Dense)             (None, 10)                210
_________________________________________________________________
dense_47 (Dense)             (None, 1)                 11
=================================================================
Total params: 3,661
Trainable params: 3,661
Non-trainable params: 0
_________________________________________________________________
```

Model History

- Model trained very well on the training data
- With the best accuracy coming around 15 epochs and lowest error at 21 epochs

In [249]:
```
pd.DataFrame(model.history.history).plot(figsize = (16,6), marker = "o")
```

Out[249]: <matplotlib.axes._subplots.AxesSubplot at 0x7fde07fce640>



In [251]:
```
##pd.DataFrame(model.history.history).to_csv('model/model_his_v1.csv')
```

Model evaluation and predicitons

- Appears that the model was only predicting clas 0 at a 89% accuracy so we need to find out why
- Issues was that the training data was not normalized
- The need to scale was necessary to make accurate predictions

In [232]:
```
predictions = (model.predict(X_test) > 0.5).astype("int32")
```

In [233]:
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.90      0.99      0.94       807
           1       0.54      0.14      0.23        98

    accuracy                           0.89       905
   macro avg       0.72      0.56      0.58       905
weighted avg       0.86      0.89      0.87       905
```

In [234]:
```
print(confusion_matrix(y_test, predictions))
```

```
[[795  12]
 [ 84  14]]
```

Project Summary

- The Goal for this project was to create a model that would assist the bank in targeting clients that had a higher probability to sign up for a Term Deposit. What was discovered was that the model had a over all accuracy of 90% with a 56%+- accuracy in saying yes to a Term Deposit.
- To improve this we will need to add more data or clients that are willing to (1) sign up for the Term Deposit
- The data, more specifically Term Deposit users was very unbalanced. with there being 4000 clients not signing up and only about 500 saying yes(1)
- looking at 12% of the data being catered to the targetof importance.
- For now we can help this bank target clients at a 56%+- accuracy in the potential that will sign up for a term Deposit, this is far better that a blind guess or taking a cold call approach.

In [215]:
```
##model.save("model/term_deposit_v1.h5")
```

In [ ]:

In [ ]:

In [ ]: