Initial Prediction Model

The original README file says:

> Often, more than one contact to the same client was required, **in order to access if the product (bank term deposit) would be (or not) subscribed**

Therefore, let's start with a simple binary classification model to predict Deposit yes/no

In [1]:
```
#import the right libraries
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
#this option just allwos us to see every column in the notebook
pd.set_option('display.max_columns', None)

#pd.get_option("display.max_columns")
```

In [3]:
```
#pull in the dataset and turn into a DataFrame
bank_main_df = pd.read_csv('./Dataset_1_Bank Marketing/bank_marketing.csv',delimiter=';')
bank_main_df.head()
```

Out[3]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	pou
0	58.0	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unl
1	44.0	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unl
2	33.0	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unl
3	47.0	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unl
4	33.0	unknown	single	unknown	no	1	no	no	NaN	5	may	198	1	-1	0	unl

In [4]:
```
bank_main_df.describe()
```

Out[4]:

	age	balance	day	duration	campaign	pdays	previous
count	43872.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.924781	1362.272058	15.806419	258.163080	2.763841	40.197828	0.580323
std	10.610835	3044.765829	8.322476	257.527812	3.098021	100.128746	2.303441
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000

In [5]:
```
#checking the options available under the "deposit" field
bank_main_df['deposit'].value_counts()
```

Out[5]:
```
no      39922
yes      5289
Name: deposit, dtype: int64
```

In [6]:
```
#replacing the yes/no categorical values with 1/0 binary digits
bank_main_df['deposit'] = [0 if (bank_main_df['deposit'][i] == 'yes') else 1 for i in range(len(bank_main_df))
```

In [7]:
```
#because we have so many cateogrical variables, we should one-hot encode them (i.e. create dummy categorical va
#we also use drop_first=True to reduce the redundant column count
bank_main_df = pd.get_dummies(bank_main_df, drop_first=True)

bank_main_df
```

Out[7]:

	age	balance	day	duration	campaign	pdays	previous	deposit	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_
0	58.0	2143	5	261	1	-1	0	1	0	0	0	1	
1	44.0	29	5	151	1	-1	0	1	0	0	0	0	
2	33.0	2	5	76	1	-1	0	1	0	1	0	0	
3	47.0	1506	5	92	1	-1	0	1	1	0	0	0	
4	33.0	1	5	198	1	-1	0	1	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
45206	51.0	825	17	977	3	-1	0	0	0	0	0	0	
45207	71.0	1729	17	456	2	-1	0	0	0	0	0	0	
45208	72.0	5715	17	1127	5	184	3	0	0	0	0	0	
45209	57.0	668	17	508	4	-1	0	1	1	0	0	0	
45210	37.0	2971	17	361	2	188	11	1	0	1	0	0	

45211 rows × 43 columns

In [8]:
```
#note that only the "age" category has null values
pd.isnull(bank_main_df).sum()
```

Out[8]:
```
age                    1339
balance                   0
day                       0
duration                  0
campaign                  0
pdays                     0
previous                  0
deposit                   0
job_blue-collar           0
job_entrepreneur          0
job_housemaid             0
job_management            0
job_retired               0
job_self-employed         0
job_services              0
job_student               0
job_technician            0
job_unemployed            0
job_unknown               0
marital_married           0
marital_single            0
education_secondary       0
education_tertiary        0
education_unknown         0
default_yes               0
housing_yes               0
loan_yes                  0
contact_telephone         0
contact_unknown           0
month_aug                 0
month_dec                 0
month_feb                 0
month_jan                 0
month_jul                 0
month_jun                 0
month_mar                 0
month_may                 0
month_nov                 0
month_oct                 0
month_sep                 0
poutcome_other            0
poutcome_success          0
poutcome_unknown          0
dtype: int64
```

In [9]:
```
#for simplification, let's just drop the nulls for now
bank_main_df = bank_main_df.dropna()
```

Setting up Logistic Regression

In [10]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
```

In [11]:
```
#set up the X matrix and y target variable
X = bank_main_df.drop(columns='deposit')
y = bank_main_df['deposit']

#split the data appropriately into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, shuffle=True)
```

In [12]:
```
#instantiate scaler and LogisticRegression model
sc = StandardScaler()
logreg = LogisticRegression()

#fit/transform the X_train and X_test datasets
X_train_sc = sc.fit_transform(X_train)
X_test_sc = sc.transform(X_test)

#train the logreg model
logreg.fit(X_train_sc, y_train)
```

Out[12]:
```
LogisticRegression()
```

In [13]:
```
#score the model
print(f"Train score: {logreg.score(X_train_sc,y_train)}")
print(f"Test score: {logreg.score(X_test_sc,y_test)}")
```

```
Train score: 0.902800390752198
Test score: 0.9003191004406625
```

In [14]:
```
#for the test dataset, make predictions for the target variable
y_preds = logreg.predict(X_test_sc)
```

In [15]:
```
print(f"Confusion matrix so we can find Type I / Type II errors:\n{confusion_matrix(y_true=y_test, y_pred=y_pre
```

```
Confusion matrix so we can find Type I / Type II errors:
[[  543  1009]
 [  303 11307]]
```

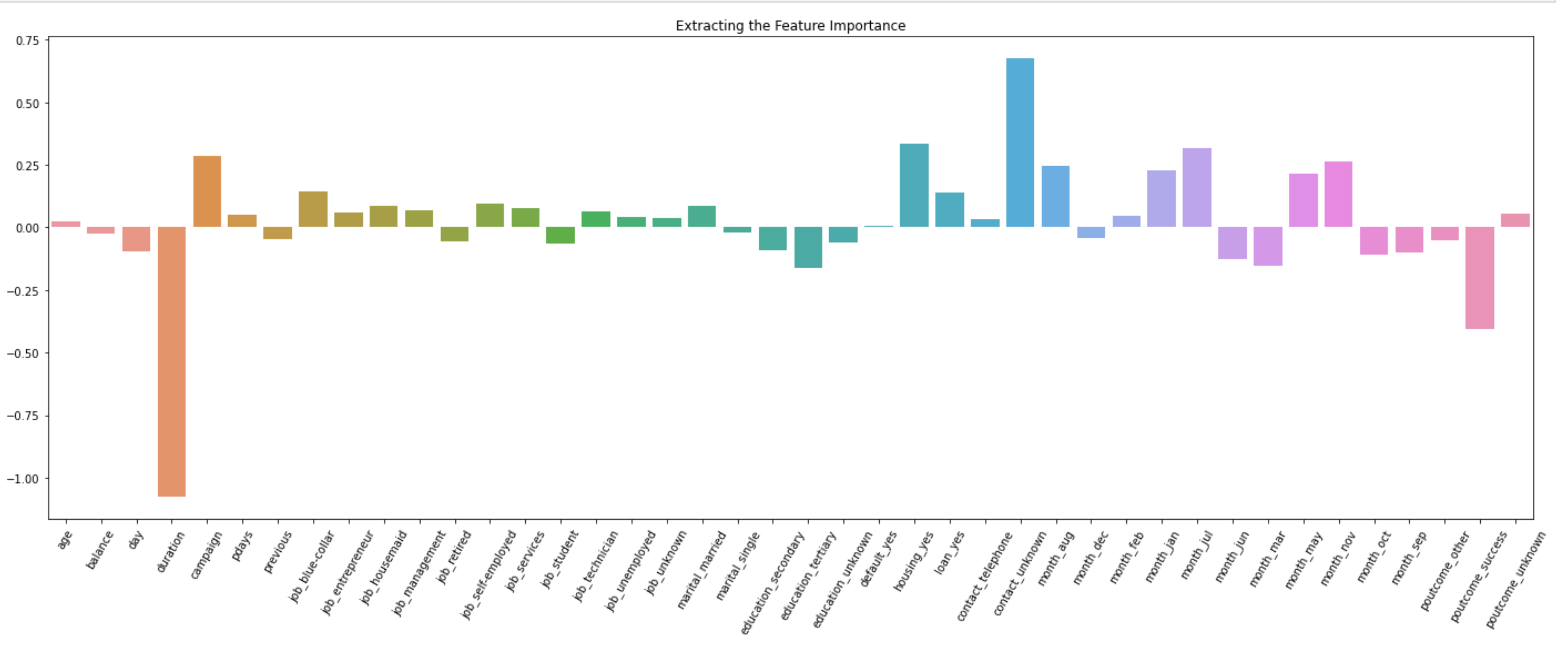
In [16]:
```
print("Here is a classification report, based on the confusion matrix")
print(classification_report(y_true=y_test,y_pred=y_preds))
```

```
Here is a classification report, based on the confusion matrix
              precision    recall  f1-score   support

           0       0.64      0.35      0.45      1552
           1       0.92      0.97      0.95     11610

    accuracy                           0.90     13162
   macro avg       0.78      0.66      0.70     13162
weighted avg       0.89      0.90      0.89     13162
```

In [18]:
```
plt.figure(figsize=(24,8))
sns.barplot(x=X.columns,y=logreg.coef_[0])
plt.xticks(rotation=60)
plt.title("Extracting the Feature Importance")
plt.savefig('Images/first_regression_features.png');
```



Further discussion for the group

- **What further refinements to the dataset should we make as part of the EDA / cleanup?**
  - Removing the *pdays* variable, for example
  - Dropping outliers
- **How might the use of other classification algorithms and scalers affect the final predictions?**
  - Algorithms like LogisticRegression, DecisionTree, RandomForest, Kneighbors, NaiveBayes, neural net, etc.
  - Scalers like StandardScaler, MinMaxScaler, RobustScaler
  - PCA (principal component analysis) to reduce dimensions
- **Playing with parameters, pipelines, gridsearches to maximize True Negatives and minimize False Negatives**
  - That is, maximize deposit==1 correct predictions and reducing deposit==0 wrong predictions
  - Even if that means accidentally overpredicting the number of true deposits, better to try a bad path than miss a potential business opportunity
- **Extending this to other predictions**
  - e.g. predicting the "default" variable, or some other classification
  - e.g. predicting a range for continuous values based on categorical values
- **Best ways to impute missing data?**

In [ ]: