This notebook utilizes a dataset from UCI Machine Learning Repository. The data comes from a Portogeuse banking institution with information where customers were targeted with a direct marketing campaign regarding their desire to subsribe to term deposits. The goal is to create a classification model to predict whether a customer would subscribe to the term deposit or not.

Moro,S., Rita,P., and Cortez,P.. (2012). Bank Marketing. UCI Machine Learning Repository. https://doi.org/10.24432/C5K306.

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

In [2]:
```
pip install ucimlrepo
```

```
Requirement already satisfied: ucimlrepo in ./opt/anaconda3/lib/python3.9
/site-packages (0.0.2)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
from ucimlrepo import fetch_ucirepo

# fetch dataset
bank_marketing = fetch_ucirepo(id=222)

# data (as pandas dataframes)
X = bank_marketing.data.features
y = bank_marketing.data.targets

# metadata
# print(bank_marketing.metadata)

# variable information
# print(bank_marketing.variables)
```

In [4]:
```
X.head()
```

Out[4]:

	age	job	marital	education	default	balance	housing	loan	contact	day_o
0	58	management	married	tertiary	no	2143	yes	no	NaN	
1	44	technician	single	secondary	no	29	yes	no	NaN	
2	33	entrepreneur	married	secondary	no	2	yes	yes	NaN	
3	47	blue-collar	married	NaN	no	1506	yes	no	NaN	
4	33	NaN	single	NaN	no	1	no	no	NaN	

In [5]:
```
X.shape
```

Out[5]:
```
(45211, 16)
```

In [6]:
```
y.head()
```

Out[6]:

	y
0	no
1	no
2	no
3	no
4	no

In [7]:

```
n_inputs = len(X)
n_yes = len(y[y['y']=='yes'])
n_no = len(y[y['y']=='no'])
perc = round(n_yes/n_inputs * 100, 2)

print(f"Number of Inputs is: {n_inputs}")
print(f"Number of Term Deposits: {n_yes}")
print(f"Number of Non-subscribers: {n_no}")
print(f"Percentage of Contacted who subscribed: {perc}%")
```

```
Number of Inputs is: 45211
Number of Term Deposits: 5289
Number of Non-subscribers: 39922
Percentage of Contacted who subscribed: 11.7%
```

From the above we can see that the majority of those contacted do not subscribe. Hence, a naive model that always suggests "no" will exhibit high levels of accuracy. However, we are intersted in exploring a model that can help identify the customers who will want to subscribe to term deposits, hence, the naive classifier is of no use.

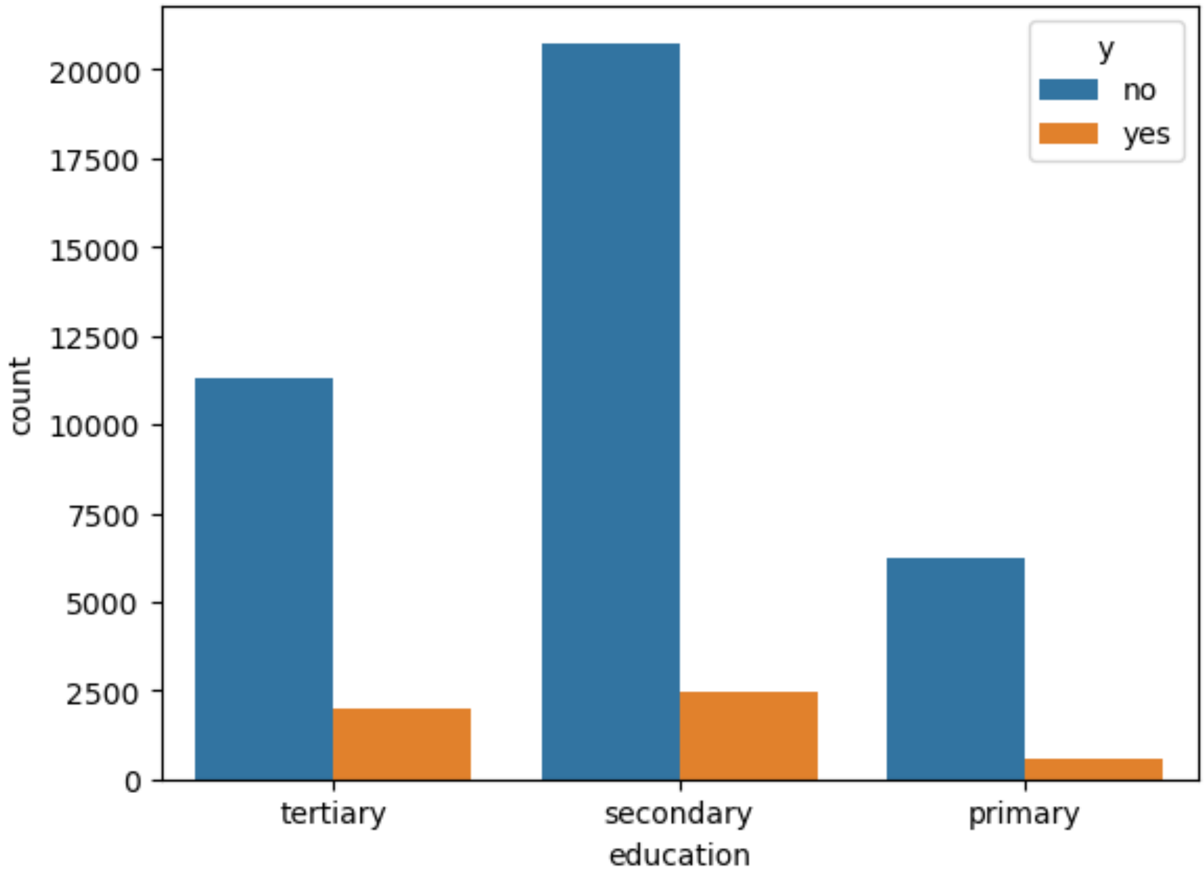
Exploring the Data

In [8]:

```
merged = pd.concat([X["education"], y["y"]], axis = 1)
sns.countplot(data=merged, x="education", hue="y" )
```

Out[8]:

```
<AxesSubplot:xlabel='education', ylabel='count'>
```



In [9]:
```
X["education"].value_counts()
```

Out[9]:
```
secondary    23202
tertiary     13301
primary       6851
Name: education, dtype: int64
```

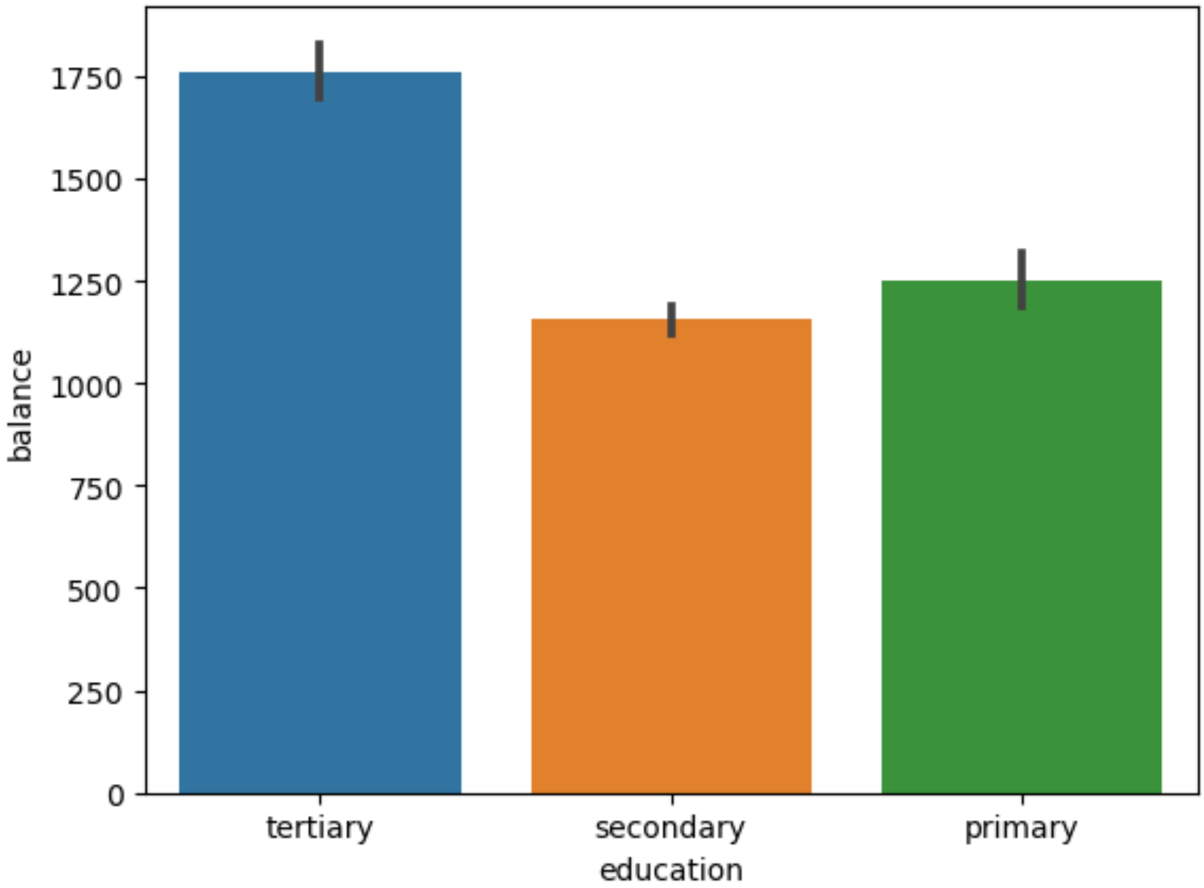
In [10]:
```
merged = pd.concat([X["education"], y["y"]], axis = 1)
merged.groupby(['education'])['y'].value_counts(normalize=True)
```

Out[10]:
```
education  y
primary    no     0.913735
           yes    0.086265
secondary  no     0.894406
           yes    0.105594
tertiary   no     0.849936
           yes    0.150064
Name: y, dtype: float64
```

As we can clearly see, the greater the level of education, the greater the percentage of subscribers. Intuitively, this is due to individuals with higher levels of education having professions that result in higher salary. As a result, they amass more savings and are able to part with more money for longer periods of time.

In [11]:
```
sns.barplot(data=X, x="education", y="balance")
```

Out[11]:
```
<AxesSubplot:xlabel='education', ylabel='balance'>
```



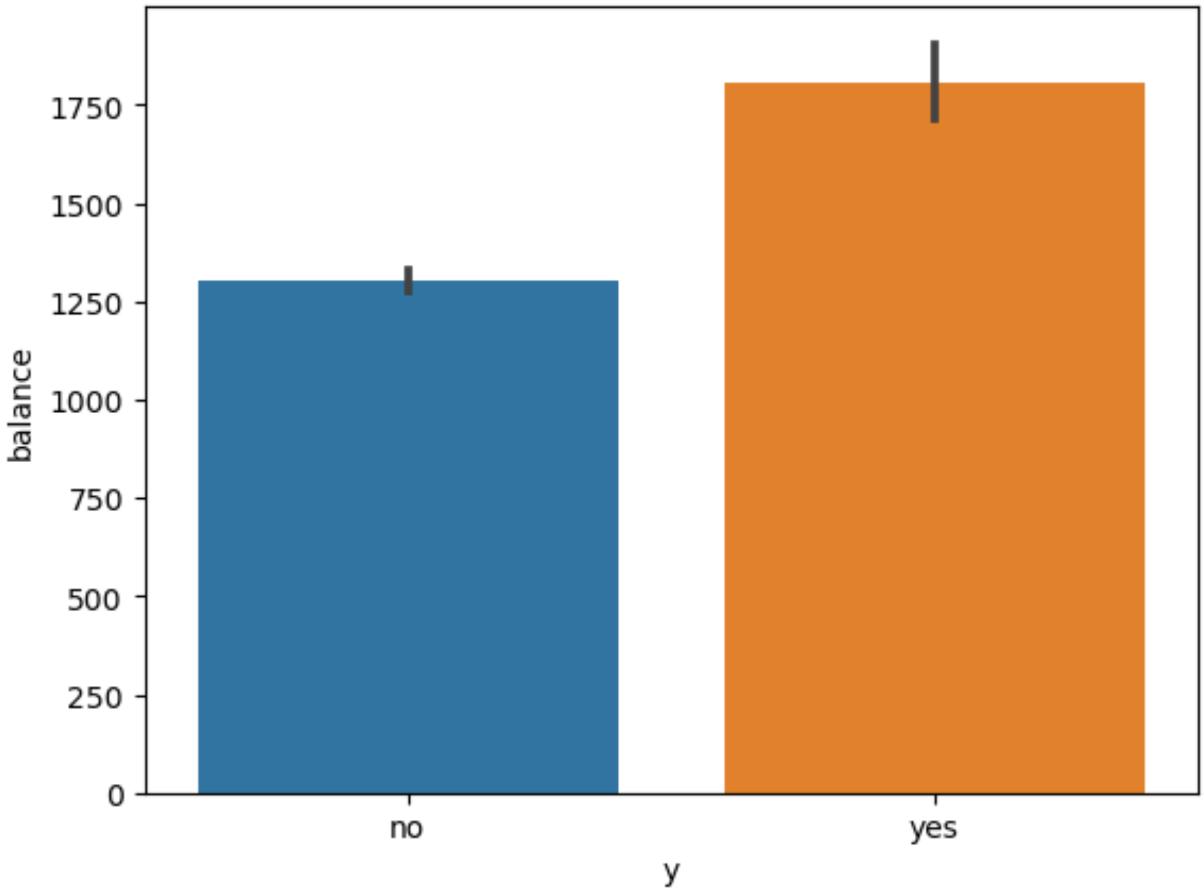
In [12]:
```
X.groupby('education')['balance'].mean()
```

Out[12]:
```
education
primary      1250.949934
secondary    1154.880786
tertiary     1758.416435
Name: balance, dtype: float64
```

Observing the outputs above, we have that this is not necessarily true as individuals with primary education have an a higher average balance than those with secondary education. It may be of more interest to look at the average balance of those who subscribed to the term deposits vs those who did not, which is clearly higher as illustrated below.

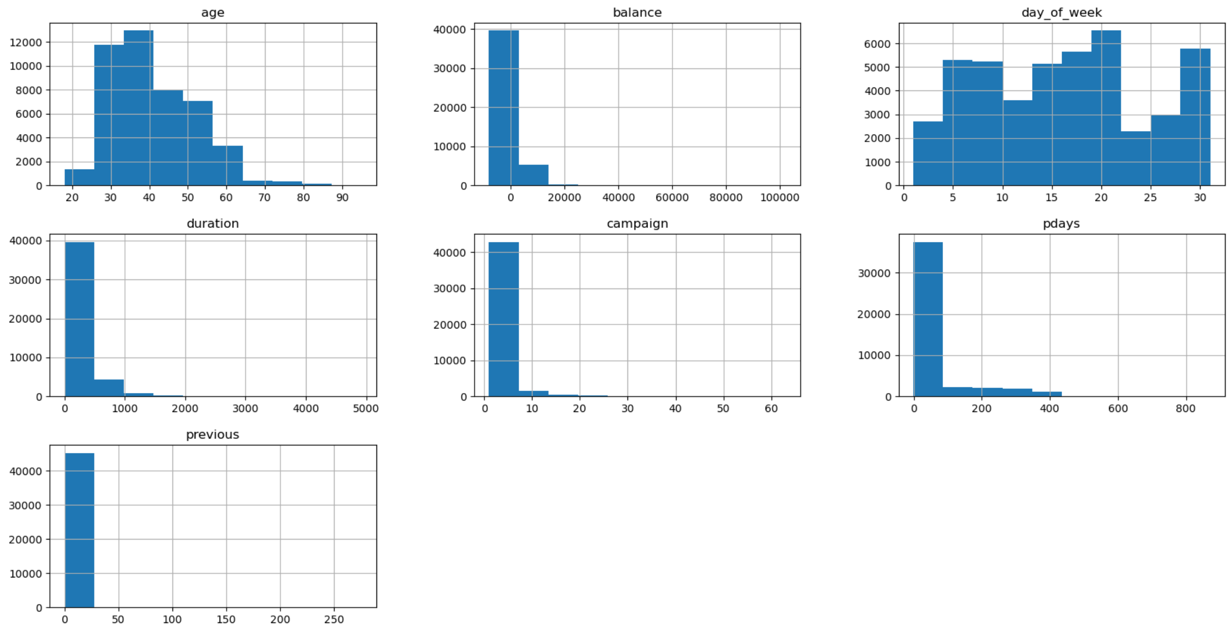
In [13]:
```
merged = pd.concat([X["balance"], y["y"]], axis = 1)
sns.barplot(data=merged, x="y", y="balance")
```

Out[13]:
```
<AxesSubplot:xlabel='y', ylabel='balance'>
```



In [14]:
```
X.hist(figsize=(20,10))
```

Out[14]:
```
array([[<AxesSubplot:title={'center':'age'}>,
        <AxesSubplot:title={'center':'balance'}>,
        <AxesSubplot:title={'center':'day_of_week'}>],
       [<AxesSubplot:title={'center':'duration'}>,
        <AxesSubplot:title={'center':'campaign'}>,
        <AxesSubplot:title={'center':'pdays'}>],
       [<AxesSubplot:title={'center':'previous'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```



Data Preprocessing

Firstly, I will begin by taking a closer look at the features with missing values

In [15]:
```
X.isna().sum()
```

Out[15]:
```
age                0
job              288
marital            0
education       1857
default            0
balance            0
housing            0
loan               0
contact        13020
day_of_week        0
month              0
duration           0
campaign           0
pdays              0
previous           0
poutcome       36959
dtype: int64
```

In [16]:
```
X['poutcome'].value_counts()
```

Out[16]:
```
failure    4901
other      1840
success    1511
Name: poutcome, dtype: int64
```

From the description of the data, it would be a sensible decision to replace all missing 'poutcome' entries with 'other' as that indicates that the outcome is unknown. I will proceed with the same intuition for all features with missing values

In [17]:
```
X['poutcome'].fillna('other', inplace=True)
```

```
/var/folders/g3/3207dkp56knd0x_rtz756wp00000gn/T/ipykernel_66194/33993901
42.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X['poutcome'].fillna('other', inplace=True)
```

In [18]:
```
X['poutcome'].value_counts()
```

Out[18]:
```
other      38799
failure     4901
success     1511
Name: poutcome, dtype: int64
```

In [19]:
```
X['contact'].value_counts()
```

Out[19]:
```
cellular     29285
telephone     2906
Name: contact, dtype: int64
```

In [20]:
```
X['contact'].fillna('no contant', inplace=True)
```

```
/var/folders/g3/3207dkp56knd0x_rtz756wp00000gn/T/ipykernel_66194/24015780
5.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X['contact'].fillna('no contant', inplace=True)
```

In [21]:
```
X['contact'].value_counts()
```

Out[21]:
```
cellular      29285
no contant    13020
telephone      2906
Name: contact, dtype: int64
```

In [22]:
```
X['education'].fillna('no infromation', inplace=True)
```

```
/var/folders/g3/3207dkp56knd0x_rtz756wp00000gn/T/ipykernel_66194/12062128
15.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X['education'].fillna('no infromation', inplace=True)
```

In [23]:
```
X['education'].value_counts()
```

Out[23]:
```
secondary         23202
tertiary          13301
primary            6851
no infromation     1857
Name: education, dtype: int64
```

In [24]:
```
X['job'].value_counts()
```

Out[24]:
```
blue-collar      9732
management       9458
technician       7597
admin.           5171
services         4154
retired          2264
self-employed    1579
entrepreneur     1487
unemployed       1303
housemaid        1240
student           938
Name: job, dtype: int64
```

In [25]:
```
X['job'].fillna('unknown', inplace=True)
```

```
/var/folders/g3/3207dkp56knd0x_rtz756wp00000gn/T/ipykernel_66194/30319533
93.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X['job'].fillna('unknown', inplace=True)
```

Following on from this, I will one-hot encode all categorical variables to allow for input into the models later on.

In [26]:
```
X = pd.get_dummies(X)
```

In [27]:
```
y['y'] = y['y'].replace({'yes': 1, 'no': 0})
```

```
/var/folders/g3/3207dkp56knd0x_rtz756wp00000gn/T/ipykernel_66194/30665292
62.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  y['y'] = y['y'].replace({'yes': 1, 'no': 0})
```

Next, I want to explore the numerical variables to identify skewness among any of them and transform them where necessary. However, I will begin by splitting the data into train and test data to prevent leakege of information from the test onto the data set.

In [28]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2

numeric = ['age', 'balance', 'duration', 'campaign', 'pdays', 'previous']

scaler = MinMaxScaler()
X_train[numeric] = scaler.fit_transform(X_train[numeric])
X_test[numeric] = scaler.fit_transform(X_test[numeric])
```

Following this, from earlier we know that the number of nonsubscribers is much larger that the number of subscribers. In a classifcation problem such as this, a model may give high accuracy just by overfitting to the majority class, similarly to what would happen under a naive classifier. To account for this, resampling can be done by either under sampling, removing some of the majority class, or by upsampling, creating synthetic data points. In this case, I will proceed with upsampling to prevent the loss of information that could be caused by undersampling. However, it is important to address the drawbacks of upsampling, if the synthetic data is too close to the original, then it may lead to overfitting. Considering, the low instances if y=1, the effect of downsampling may be too large and hence, upsampling may be more fitting to the goal.

In [29]:
```
from sklearn.utils import resample

train_data = pd.concat([X_train, y_train], axis=1)
train_data.head()

negative = train_data[train_data.y==0]
positive = train_data[train_data.y==1]

upsampled = resample(positive, replace=True, n_samples=len(negative), ran

train_upsampled = pd.concat([negative, upsampled])

y_train_upsampled = train_upsampled['y']
train_upsampled.drop('y', axis=1, inplace=True)
```

Performance Metric

In this problem, the goal is to predict the customers who would subscribe to the term deposit and so we are interested in the accuracy of our model. As such, several performance metrics present themselves including confusion matrices, log-loss and ROC AUC. As contacting large number of customer is a costly operation for most insititutions, both financially and in opportunity cost. Hence, we want to reduce the uncertainty caused by our model. Therefore, we will use log-loss as our performance metric. This is becuase it penalises classifiers that are confident about the incorrect predicions and selects models that are confident about the correct label. Log-loss can be defined as:

$$\text{log-loss} = -\frac{1}{N}\sum_{i=1}^{N} y_i log(p_i) + (1-y_i) log(1-p_i)$$

The interpretation of this is simple. Probabilities that are exactly equal to the correct class do not contribute to the mean log-loss of a model. However, as the probability gets furher away from the true value, the contribution to the mean log-loss increases. Therefore, we get a metric that informs us of how certain our model is.

Model Evaluation and Selection

To perform this classification problem, four different popular classification algorithms will be explored to decide the most suitable for our problem.

SVM

SVM can be used for various supervised modelling scenarios including both regression and classification problem. SVM works by separating the data using a hyperplane. Under hard-svm conditions, the data must be perfectly linearly separable. However, soft-svm introduces slack variables that allow for the application of soft-svm to non-linearly separable data. It is a popular algorithm for many reasons including its utilisation of a kernel to efficiently work with high dimensional data. However, it can be sensitive to noise, specifically in classificaion problems.

Decision Tree

Decision trees use a divide and conquer recursive process beginning with the whole dataset at the root node and proceeds to partition the data. Splitting decision and criteria are decided by the algorithm using the concept of entropy. The split that has the lowest entropy is the one decided on by the algorithm. One main advantage of this method is that it does not rely on the relationship between the features and the target variable being linear. However, they are prone to overfitting. This can be solved using

Random Forests

Random forests can help build a more robust predictor than decision trees. Random forests work by randomly picking a subset of variables to consider at each splitting decision. They belong to a class of methods called ensemble methods as you train multiple trees and combine their predictions to make final predictions. Moreover, each tree is trained on a different subset of data obtained by bootstrapping. As a result, we have a more robust predictor than a simple decision tree.

In [30]:
```
from sklearn.metrics import log_loss
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [31]:

```
models = {'SVC': SVC(random_state=1), 'Decision Tree': DecisionTreeClassi

model_results = []
model_names = []

for name, model in models.items():
    a = model.fit(train_upsampled, y_train_upsampled)
    predicted = a.predict(X_test)
    score = log_loss(y_test, predicted)
    model_results.append(score)
    model_names.append(name)

    train_results = pd.DataFrame([model_names, model_results])
    train_results = train_results.transpose()
    train_results = train_results.rename(columns={0: 'Models', 1: "log-lo
print(train_results)
```

```
          Models  log-loss
0            SVC  6.710814
1  Decision Tree  5.270814
2  Random Forest  4.033309
```

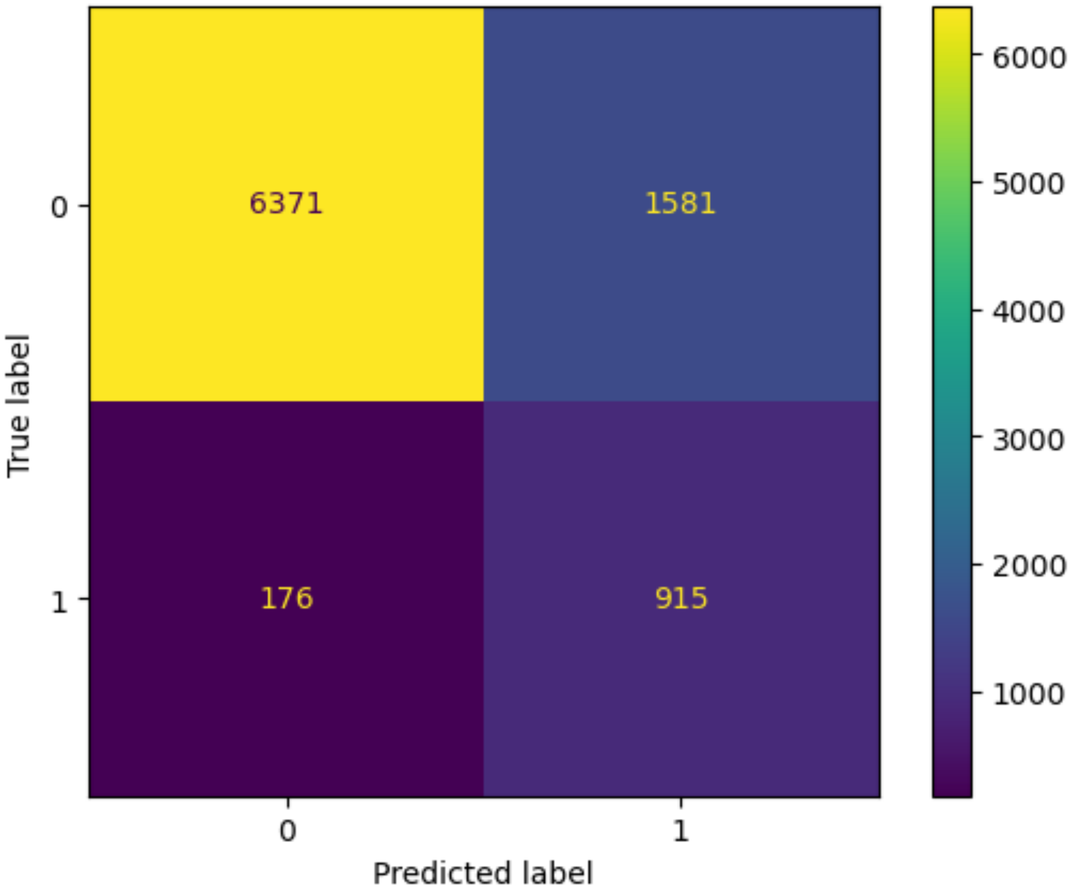
One drawback of using log-loss as our performance metric is that we cannot directly interperet the raw values, However, here it is clear to see that the Random Forests have the lowest loss. However, to fully understand the accuracy of each model, I will add another perfomance metric, confusion matrices, to gain a greater understanding of the performance of the models.

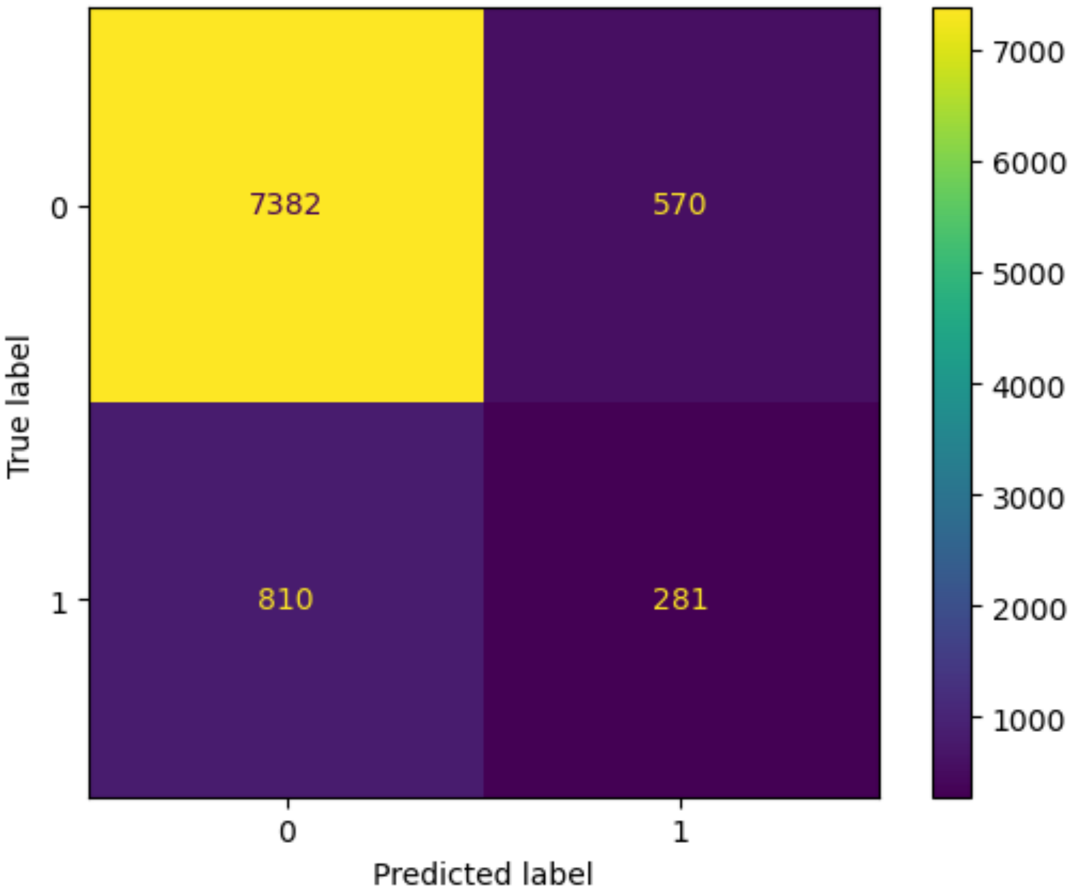
In [32]:

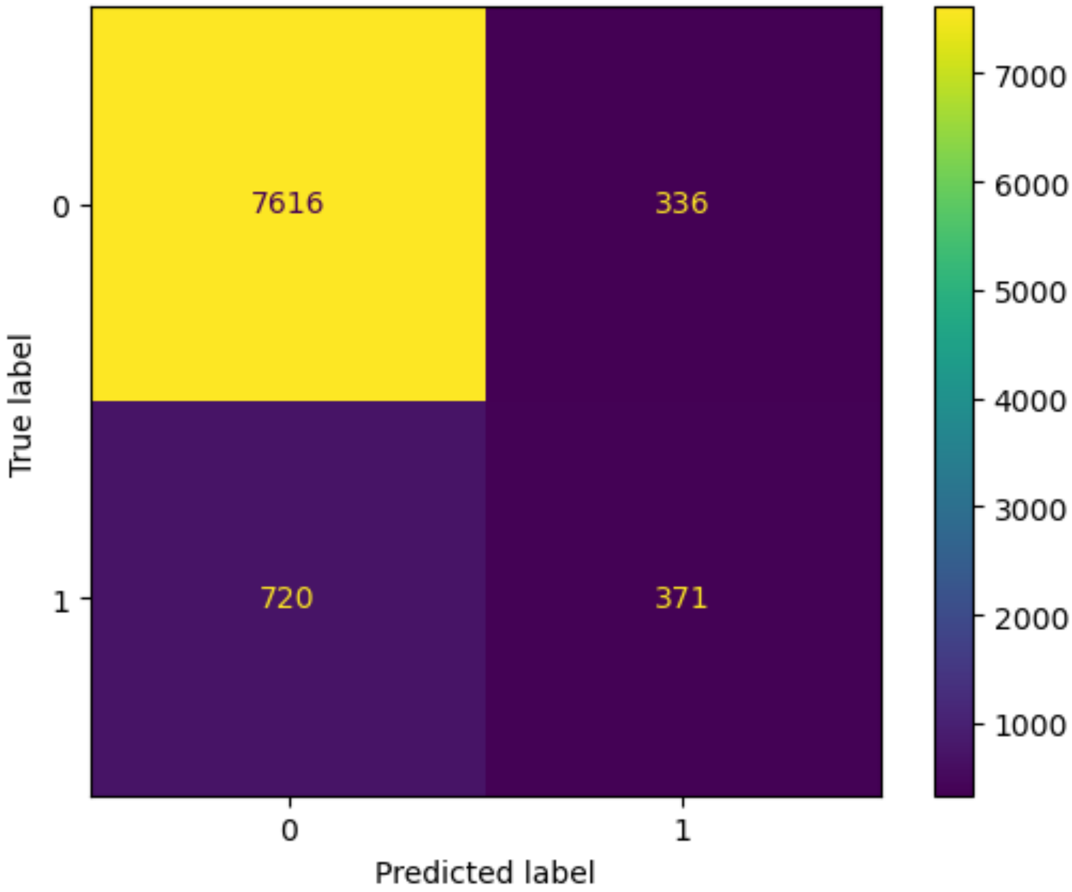
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

for name, model in models.items():
    predicted = model.predict(X_test)
    confusion = confusion_matrix(y_test, predicted, labels=model.classes_
    disp = ConfusionMatrixDisplay(confusion_matrix=confusion, display_lab
    disp.plot()
    plt.show()

    #plt.figure
    #sns.heatmap(confusion, annot=True, cmap="crest", square=True)
    #plt.ylabel("True Label")
    #plt.xlabel("Predicted Label")
    #plt.title(f"{name}")
```


6371
1581
176
915
True label
Predicted label
0
1
0
1
6000
5000
4000
3000
2000
1000


7382
570
810
281
True label
Predicted label
0
1
0
1
7000
6000
5000
4000
3000
2000
1000



Despite having the highest log loss score, an analysis of the confusion matrices indicates that it may be the best choice in our specific context. As the bank is trying to identify the customers willing to subscribe to the term deposits, SVM performs the best in this context as it correctly predicted the largest number of subscribers. It also minimises type 2 errors, which is important in this context for the bank to maximise the number of subsribers. Its worse performance in log-loss comes from its higher porbability of type 1 error. However, identifying non-subscribers is not as important of an objective in this context. Hence, we can conclude that SVM is the best choice for the banking institution to use.

Noteboook Evaluation

What went well?

1. Ouputted a model that predicts with fairly high recall which is important in the context of our dataset.

2. Explored different methods of managing imbalanced datasets.

3. Gained an improved understanding of the particular algorithms used and explored different perfomance metrics for classification problems

What could be improved?

1. The dataset used was very clean, which represents an unrealistic standpoint in the real world. In future projects, a more raw dataset should be used to mimic real world scenarios experienced in data science and ML.

2. No feature selection. A more rigorous approach to modelling involves well thought through feature selection which I have not done.

3. Hyperparameter tuning. The sklearn algorithms used above all have numerous parameters that can be altered to improve and tailor the models. In this context, as SVM is computationally slow, performing hyperparameter tuning was not optimal. Howver, in the future, I want to explore this further.

In [ ]:

```

```