4. Deployment

```
# Basic Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings(action='ignore')

# Preprocessing Libraries
from sklearn.preprocessing import RobustScaler

# Model training libraries
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import RandomizedSearchCV

# Both Undersampling & Oversampling
from sklearn.linear_model import LogisticRegression

# For checking acuracy
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
```

In the beginning of the code, we will be importing related libraries that is the basic libraries, preprocessing libraries, model training libraries, under sampling and oversampling libraries and lastly libraries for checking accuracy.

```
data = pd.read_csv('onlinefraud.csv')
print(data.shape)
data.head()
```

✓ 9.3s

```
(6362620, 11)
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	0	0
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	0	0
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	1	0
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	1	0
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	0	0

In this line of code, it reads CSV file named 'onlinefraud.csv' using the pandas library and assigns the result data to a variable called 'Data'. We can see that the data was displayed the first five rows of the 'Data' Data Frame.

```
data.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column          Dtype
---  ------          -----
 0   step            int64
 1   type            object
 2   amount          float64
 3   nameOrig        object
 4   oldbalanceOrg   float64
 5   newbalanceOrig  float64
 6   nameDest        object
 7   oldbalanceDest  float64
 8   newbalanceDest  float64
 9   isFraud         int64
 10  isFlaggedFraud  int64
dtypes: float64(5), int64(3), object(3)
memory usage: 534.0+ MB
```

The code executes the data type for each of the features

```
# Checking if there are any null values in our dataset
data.isnull().sum()
```

✓ 0.6s

```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
```

Data.isnull().sum() checks if there are any null values in our dataset

```
data[data['isFraud']==1]
```

✓ 0.0s

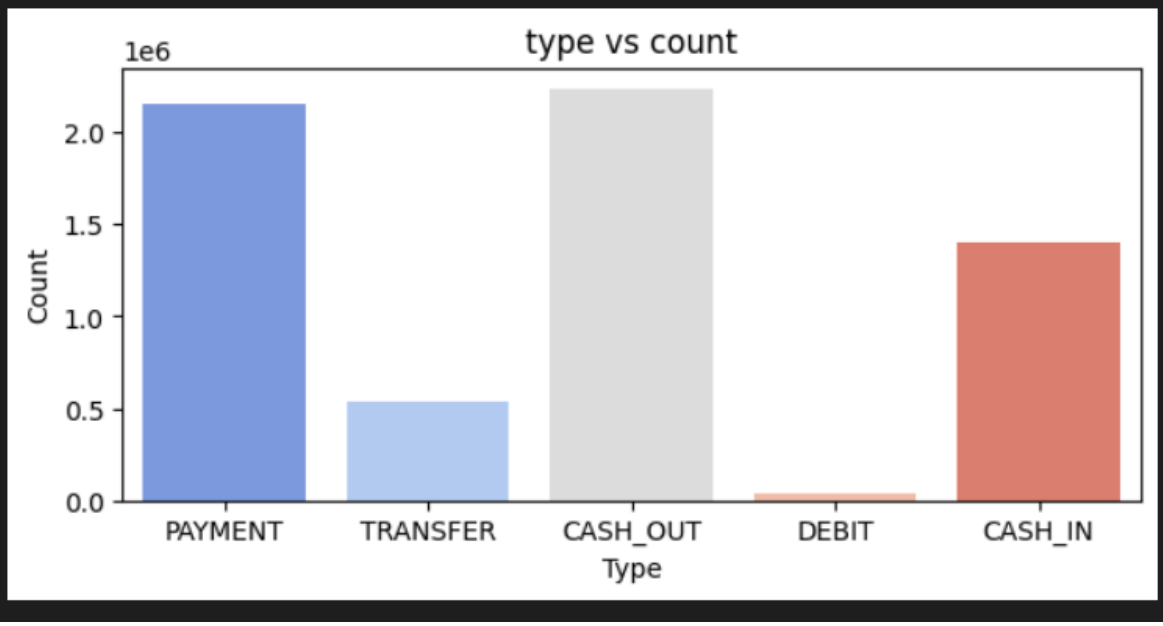
	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
2	1	TRANSFER	181.00	C1305486145	181.00	0.0	C553264065	0.00	0.00	1	0
3	1	CASH_OUT	181.00	C840083671	181.00	0.0	C38997010	21182.00	0.00	1	0
251	1	TRANSFER	2806.00	C1420196421	2806.00	0.0	C972765878	0.00	0.00	1	0
252	1	CASH_OUT	2806.00	C2101527076	2806.00	0.0	C1007251739	26202.00	0.00	1	0
680	1	TRANSFER	20128.00	C137533655	20128.00	0.0	C1848415041	0.00	0.00	1	0
...	...	...	...	...	...	...	...	...	...	...	...
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.0	C776919290	0.00	339682.13	1	0
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.0	C1881841831	0.00	0.00	1	0
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.0	C1365125890	68488.84	6379898.11	1	0
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.0	C2080388513	0.00	0.00	1	0
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.0	C873221189	6510099.11	7360101.63	1	0

8213 rows × 11 columns

This code filters the 'data' Data Frame to only include rows where the value in the 'isFraud' column is equal to 1. By passing this boolean mask as an index to the 'data' Data Frame, data[data['isFraud']==1] returns a new Data Frame that contains only the rows where 'isFraud' is equal to 1. This effectively filters the dataset to show only the rows associated with fraudulent transactions or activities.

```
# Countplot of 'type'
plt.figure(figsize=(7,3))
plt.title('type vs count')
sns.countplot(data=data,x='type',palette='coolwarm')
plt.xlabel('Type')
plt.ylabel('Count')
plt.show()
```

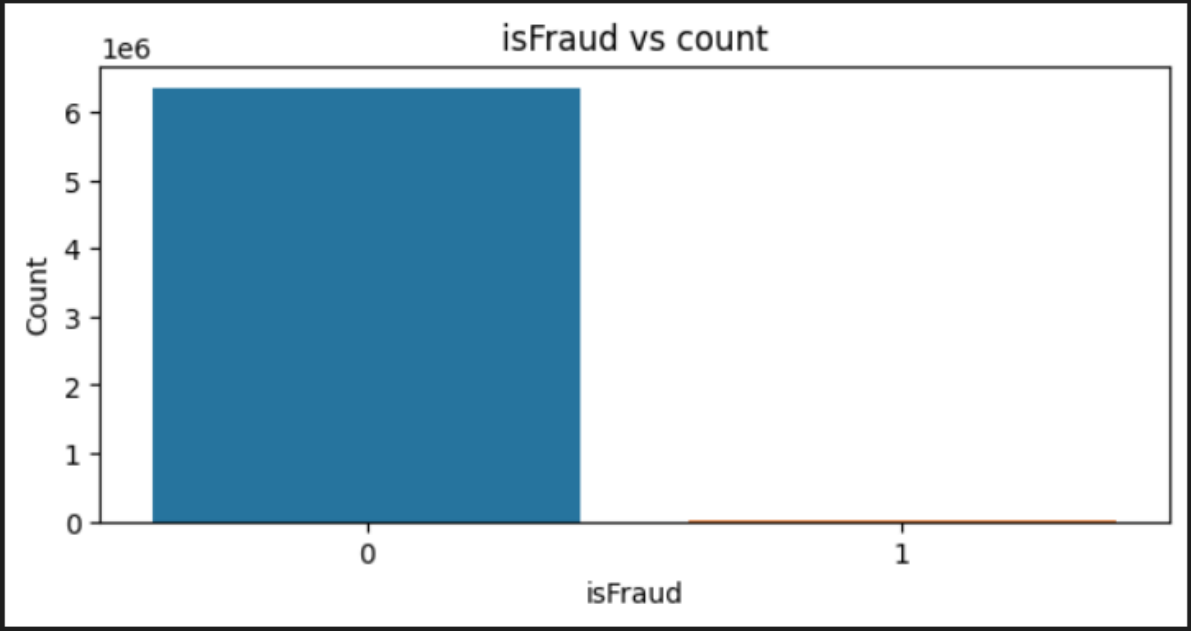
✓ 2.3s



This code creates a countplot using the seaborn library to visualize the distribution of the 'type' column in the 'data' Data Frame. We saw that the cash_out type has the highest count and Debit type has the lowest count.

```
# Countplot of 'isFraud'
plt.figure(figsize=(7,3))
plt.title('isFraud vs count')
sns.countplot(data=data,x='isFraud')
plt.xlabel('isFraud')
plt.ylabel('Count')
plt.show()
```

✓ 0.7s



This code creates a countplot using the seaborn library to visualize the distribution of the 'isFraud' column in the 'data' DataFrame. This plot will show the count of fraud and non-fraud instances in the 'isFraud' column of the dataset.

```
# the dataset is imbalanced
data['isFraud'].value_counts()
```
✓ 0.0s

```
isFraud
0    6354407
1       8213
Name: count, dtype: int64
```

From the code above, we got to figure out the number of datasets that are 'fraud' and 'not fraud'. As we can see, the data is imbalanced since the amount of data that is fraud (8213) is so little compared to data that is not fraud (6 354 407).

```
# Check percentage of each category in isFraud column(target column)
print("No Frauds:",data['isFraud'].value_counts()[0]/len(data['isFraud'])*100)
print("Frauds:",data['isFraud'].value_counts()[1]/len(data['isFraud'])*100)
```
✓ 0.1s

```
No Frauds: 99.87091795518198
Frauds: 0.12908204481801522
```

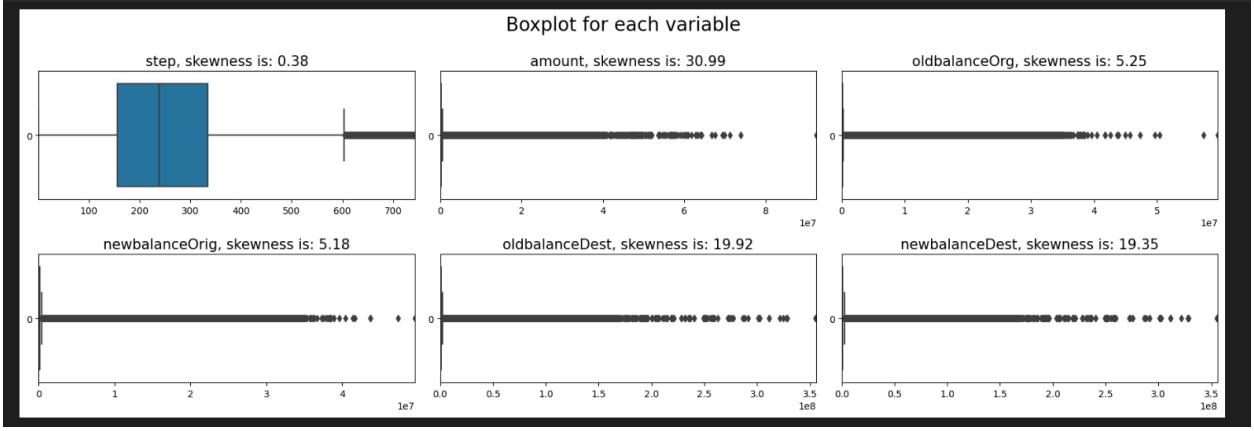
This code has the same function as the previous code, it only outputs the number of datasets that are 'fraud' and 'not fraud' in percentage form.

```
numerical=['step','amount','oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest']
```
✓ 0.0s

This code is for selecting all the columns that have numerical values for us to check the outliers and skewness of the values and insert it into a list named numerical.

```python
# Boxplot for each variable in numerical list
def boxplots_visual(data,column):
    fig, ax = plt.subplots(2,3,figsize=(18,6))
    fig.suptitle('Boxplot for each variable',y=1, size=20)
    ax=ax.flatten()
    for i,feature in enumerate(column):
        sns.boxplot(data=data[feature],ax=ax[i], orient='h')
        ax[i].set_title(feature+ ', skewness is: '+str(round(data[feature].skew(axis = 0, skipna = True),2)),fontsize=15)
        ax[i].set_xlim([min(data[feature]), max(data[feature])])
boxplots_visual(data=data,column=numerical)
plt.tight_layout()
```

✓ 19.1s Python



This code is for creating boxplots which use to check outliers in each column in the numerical list from the previous code. From the boxplot that we have created, we can see the skewness of the values in the graph.

```
# Dropping columns that are not needed
data.drop(['nameOrig','nameDest','isFlaggedFraud'],axis=1,inplace=True)
```
✓ 0.1s

This code is to drop all the data that we do not need.

```
# Applying one hot encoding on type column
data=pd.get_dummies(data=data,columns=['type'],drop_first=True)
data.head()
```
✓ 0.8s

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	type_CASH_OUT	type_DEBIT	type_PAYMENT	type_TRANSFER
0	1	9839.64	170136.0	160296.36	0.0	0.0	0	False	False	True	False
1	1	1864.28	21249.0	19384.72	0.0	0.0	0	False	False	True	False
2	1	181.00	181.0	0.00	0.0	0.0	1	False	False	False	True
3	1	181.00	181.0	0.00	21182.0	0.0	1	True	False	False	False
4	1	11668.14	41554.0	29885.86	0.0	0.0	0	False	False	True	False

This code is utilizing the one hot encoding to represent the type of transaction method that the users have used ie. cash out, debit, payment, and transfer by changing it from string to one hot vectors as a method of classification into fixed classes of the type of payment mode.

```
# We are using RobustScaler to scale down the numerical features as RobustScaler is less prone to outliers
scale=RobustScaler()
for feature in numerical:
    data[feature]=scale.fit_transform(data[feature].values.reshape(-1, 1))
data.head()
```
✓ 1.3s

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	type_CASH_OUT	type_DEBIT	type_PAYMENT	type_TRANSFER
0	-1.329609	-0.332932	1.452991	1.111175	-0.140722	-0.193057	0	False	False	True	False
1	-1.329609	-0.373762	0.065610	0.134375	-0.140722	-0.193057	0	False	False	True	False
2	-1.329609	-0.382380	-0.130708	0.000000	-0.140722	-0.193057	1	False	False	False	True
3	-1.329609	-0.382380	-0.130708	0.000000	-0.118260	-0.193057	1	True	False	False	False
4	-1.329609	-0.323571	0.254820	0.207169	-0.140722	-0.193057	0	False	False	True	False

The code above shows that we use RobustScaler to standardise our data. The way it works is by removing and replacing the outliers. It replaces the outliers with the mean value, or the average of the class that it wants to take. Our flaw is that we could not use our data that we have obtained as raw as it is due to the presence of outliers. Hence is why we use RobustScaler to make our data more presentable and also helps to class our data by range. This is so that it is easier for us to implement our data in our model.

```
# Splitting our data into independent and dependent features
x=data.drop('isFraud',axis=1)
y=data['isFraud']
```
✓ 0.2s

Here we will be splitting the datasets into two parts x and y, where x contains all the columns except for ‘isFraud’, and y contains only the ‘isFraud’ column.

```
x.columns
```
✓ 0.0s

```
Index(['step', 'amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest',
       'newbalanceDest', 'type_CASH_OUT', 'type_DEBIT', 'type_PAYMENT',
       'type_TRANSFER'],
      dtype='object')
```

We use this line of code to check the columns that are contained inside the variable x and we can see that the column ‘isFraud’ has been dropped.

```
data[data['isFraud']==1]
```

✓ 0.0s

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	type_CASH_OUT	type_DEBIT	type_PAYMENT	type_TRANSFER
2	-1.329609	-0.382380	-0.130708	0.0	-0.140722	-0.193057	1	False	False	False	True
3	-1.329609	-0.382380	-0.130708	0.0	-0.118260	-0.193057	1	True	False	False	False
251	-1.329609	-0.368941	-0.106248	0.0	-0.140722	-0.193057	1	False	False	False	True
252	-1.329609	-0.368941	-0.106248	0.0	-0.112937	-0.193057	1	True	False	False	False
680	-1.329609	-0.280261	0.055165	0.0	-0.140722	-0.193057	1	False	False	False	True
...	...	...	...	...	...	...	...	...	...	...	...
6362615	2.815642	1.355693	3.032881	0.0	-0.140722	0.112438	1	True	False	False	False
6362616	2.815642	31.927899	58.679504	0.0	-0.140722	-0.193057	1	False	False	False	True
6362617	2.815642	31.927899	58.679504	0.0	-0.068096	5.544730	1	True	False	False	False
6362618	2.815642	3.968274	7.788223	0.0	-0.140722	-0.193057	1	False	False	False	True
6362619	2.815642	3.968274	7.788223	0.0	6.762614	6.426280	1	True	False	False	False

Here we want to display the filtered datasets by selecting rows where 'isFraud' column value is 1, indicating fraud.

```
# Feature Importance
from sklearn.ensemble import ExtraTreesRegressor
model = ExtraTreesRegressor()
model.fit(x,y)
print(model.feature_importances_)
```

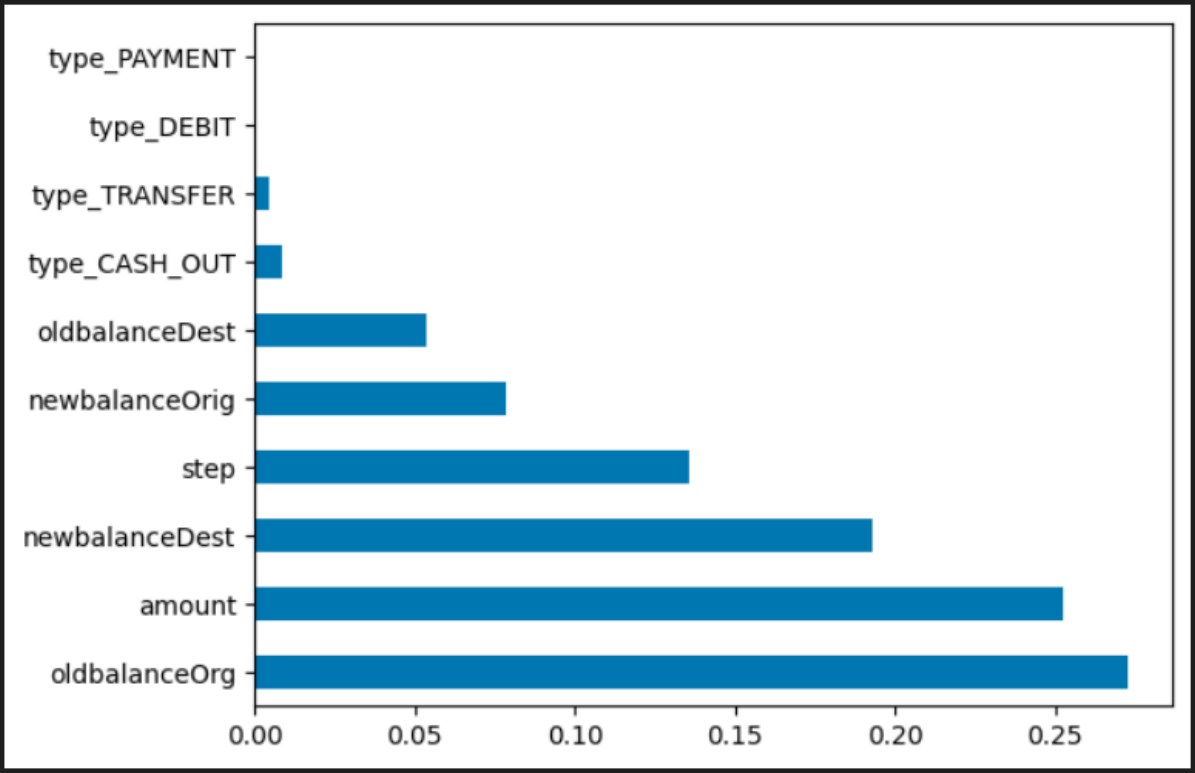
✓ 7m 17.7s

```
[0.13598966 0.25262472 0.27306121 0.07858436 0.05395581 0.19290628
 0.00855305 0.         0.         0.00432491]
```

This part of the code imports the ExtraTreeRegressor class from the skleanr.ensemble module. We then make an instance of the class and fitted the model with x and y data to train it. Then, we printed the feature importance values of each feature so that we can see the distribution of contributions by each feature.

```
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=x.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```

✓ 0.1s



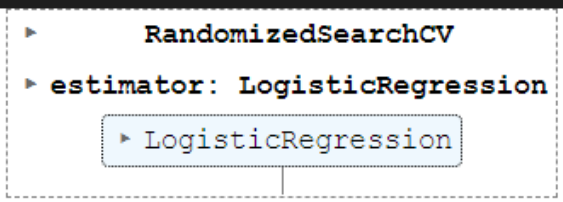
We then store the feature importance attributes as pandas Series with feature names as the index in a variable name feat_importances. Now we want to make a horizontal bar plot with the feature_importances, we use nlargest method with parameter 10 to retrieve top 10 values from the feature Series we made based on their magnitude in this case being their importance values.

```
# Doing train_test_split
X_train,X_test,y_train,y_test=train_test_split(x,y,train_size=0.7)
# Applying StratifiedKFold
skf=StratifiedKFold(n_splits=3, shuffle=False, random_state=None)
```
✓ 1.5s

Now we will be implementing the train test split technique to train our data. We split the x and y data into training (X_train, y_train) and testing (X_test, y_test) sets. Then, we will be making an instance of the StratifiedKFold class and pass in the necessary parameters.

```
model1=LogisticRegression()
param={'C':10.0 **np.arange(-1,2)}
lrs=RandomizedSearchCV(model1,param,cv=skf,n_jobs=-1,scoring='accuracy')
lrs.fit(X_train,y_train)
```

✓ 1m 55.0s



We then proceed with making an instance of the LogisticRegression class as we are going to be implementing a randomized search for hyperparameters tuning of a Logistic Regression model using cross-validation. We need to specify the hyperparameters we are going to used, so we initialized variable param with a range of 'C' value from 0.1 to 10 so the hyperparameter tuning process can explore different magnitude of 'C'. We then make an instance of the RandomizedSearchCV class we are going to be using and passing in the parameters necessary. Finally, we then fitted our randomize search model with the X_train and y_train data.

```
y_pred=lrs.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

✓ 3.3s

```
[[1906237     130]
 [   1252    1167]]
0.9992759796016945
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1906367
           1       0.90      0.48      0.63      2419

    accuracy                           1.00   1908786
   macro avg       0.95      0.74      0.81   1908786
weighted avg       1.00      1.00      1.00   1908786
```

After finishing training our model with the dataset, we are going to use the predict method on our trained model to make a prediction on the test dataset (X_test) and store the prediction in y_pred. For us to analyse the results of our data analytics process, we will print three things that will take y_test and y_pred as its parameters: confusin matrix, accuracy score and classification report.