BANK MARKETING ANALYSIS

Columns: 'age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', ' month', 'duration', 'campaign', 'pdays' , 'previous', 'poutcome', 'y'

```
data = pd.read_csv("bank.csv" , sep = ";")
```

```
data.head()
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	79	1	-1	0	unknown	no
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	1	339	4	failure	no
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	185	1	330	1	failure	no
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	4	-1	0	unknown	no
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	1	-1	0	unknown	no

```
data.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
```

- **It is a binary class classification problem.**

DATA CLEANING:

```
data.isna().sum()
```

```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
dtype: int64
```

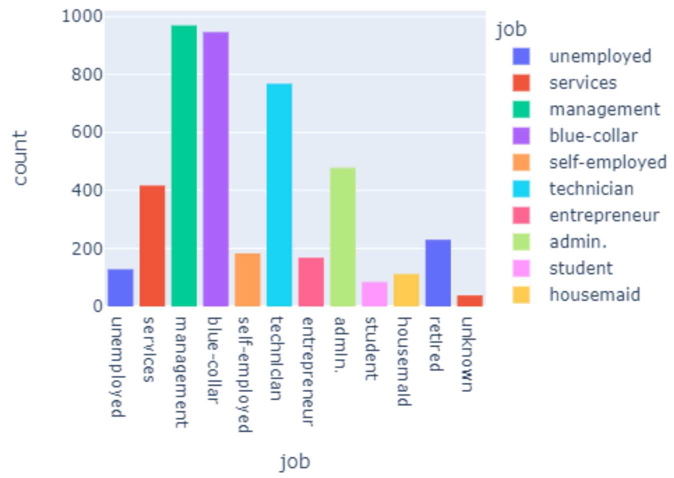
- **No Missing Values**

```
In [6]:
column_list = data.columns.values.tolist()

for col in column_list:
    print('\033[1m  {}  \033[0m'.format(col))
    print(data[col].unique())
    print("Number of total unique value for {} feature is {}:".format(col , len(data[col].unique()) ))
    print('\n')
```

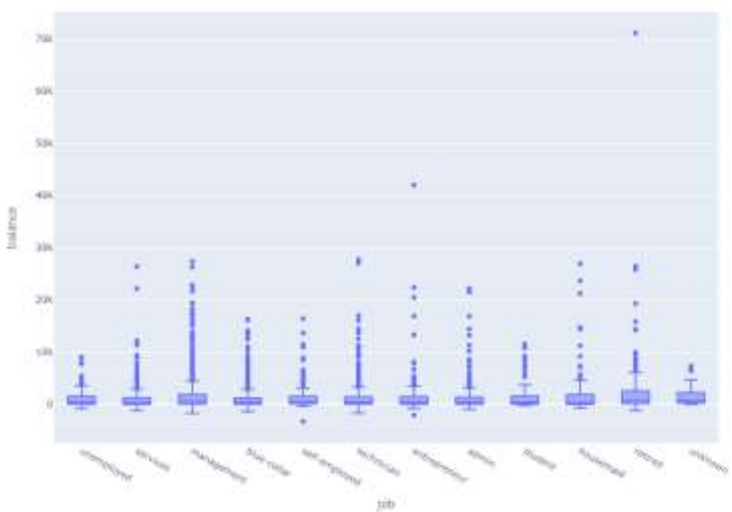
```
 age
[30 33 35 59 36 39 41 43 20 31 40 56 37 25 38 42 44 26 55 67 53 68 32 49
 78 23 52 34 61 45 48 57 54 63 51 29 50 27 60 28 21 58 22 46 24 77 75 47
 70 65 64 62 66 19 81 83 80 71 72 69 79 73 86 74 76 87 84]
Number of total unique value for age feature is 67:


 job
['unemployed' 'services' 'management' 'blue-collar' 'self-employed'
 'technician' 'entrepreneur' 'admin.' 'student' 'housemaid' 'retired'
 'unknown']
Number of total unique value for job feature is 12:
```

- **No as such removable unique values.**

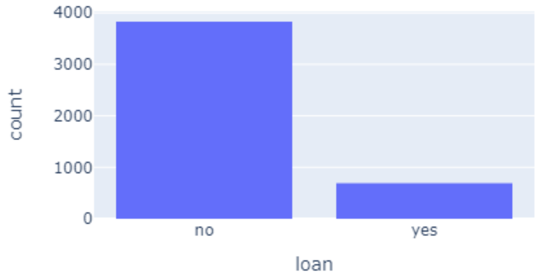
EXPLORATORY DATA ANALYSIS:

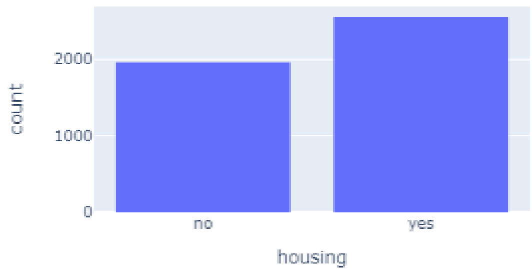


- **Retired, management, technician sector has the people with highest balance. There are outliers in retired and entrepreneur sector for balance.**

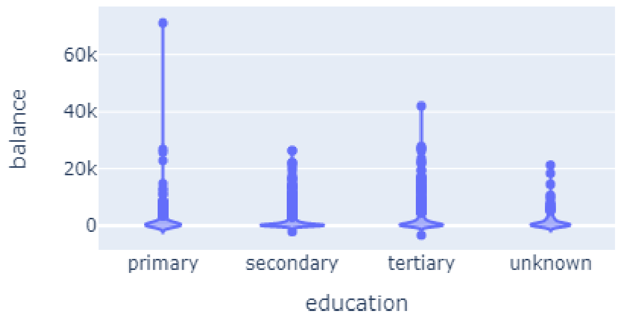


- **There are lot of people belonging to the management sector, also very few belonging to the retired sector. Yet, the retired sector has good balance.**

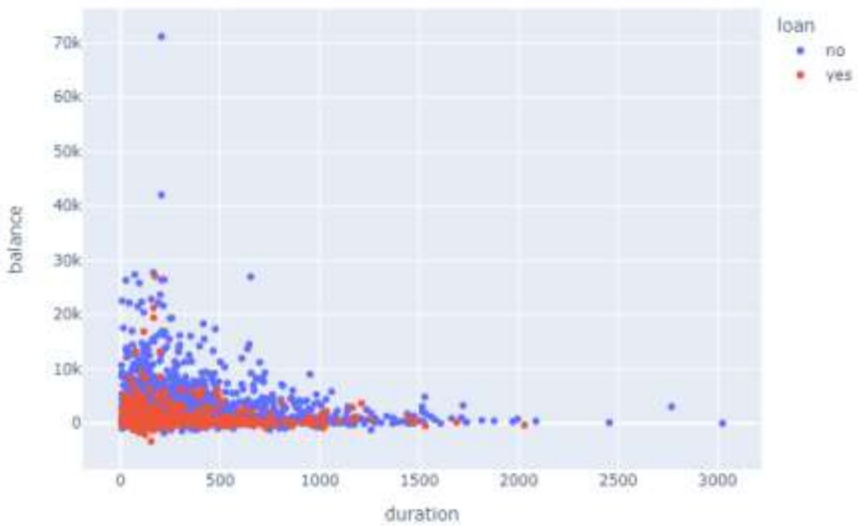




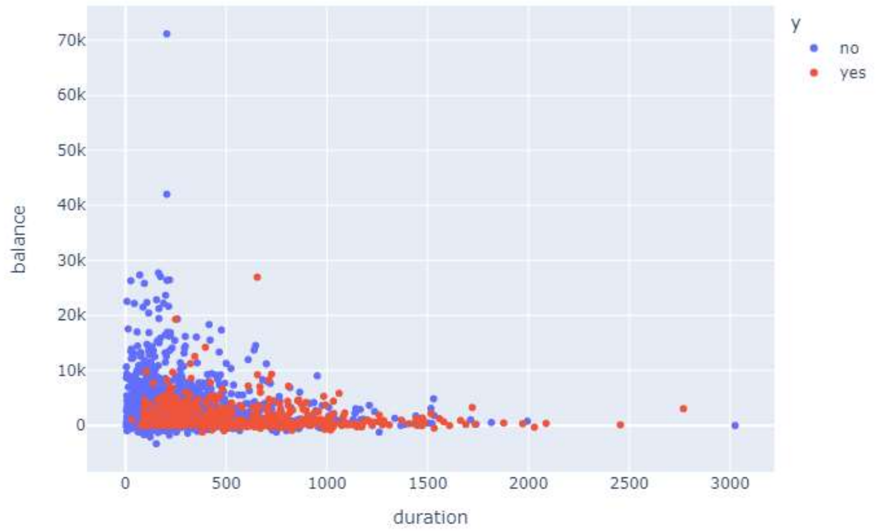
- **People are taking more housing loan than personal loan.**



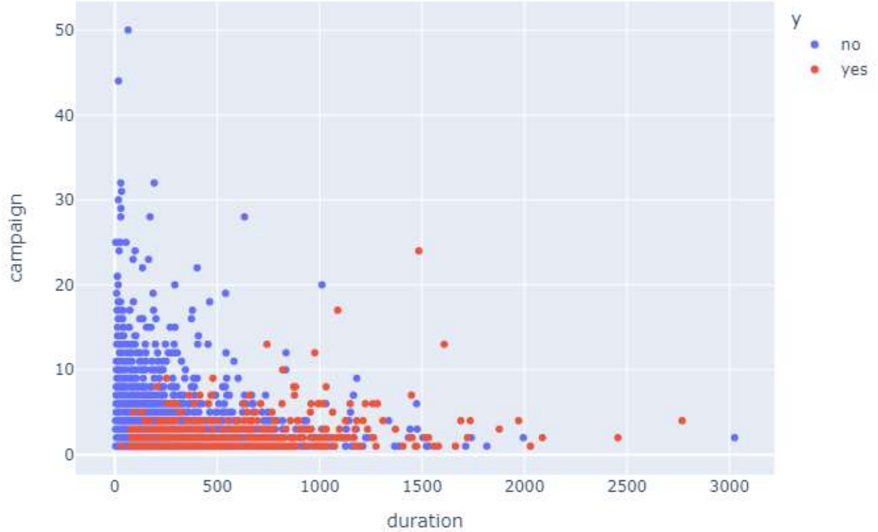
- **Education of tertiary sector has highest balance.**



- **People who have not took up a loan and have low balance have higher duration, maybe because they want to take a loan.**



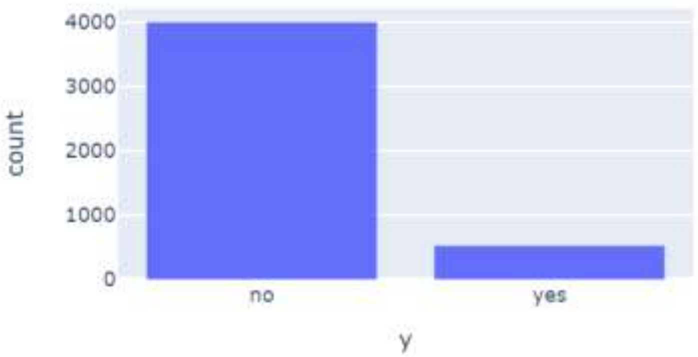
- **Lot of people with low bank balance are very much interested in term deposit and is evident from the duration feature. Basically, there are lot of people who have loan and low bank balance have term deposited.**



- **People with a smaller number of calls and less bank balance are also interested in term deposit.**
- **Maybe calling a person again and again did not really help. But rather engaging a person with least number of calls helped the company bag an investor.**

MODEL BUILDING:

```
fig = px.histogram(data, x = "y" , width = 500 , height = 300)
fig.show()

```



- **The dataset is imbalanced**

```
oversample = SMOTE()
a, b= oversample.fit_resample(x_, data["y"])
```

```
b
```

```
0       0
1       0
2       0
3       0
4       0
       ..
7995    1
7996    1
7997    1
7998    1
7999    1
Name: y, Length: 8000, dtype: int32
```

```
counter = Counter(b)
counter
```

```
Counter({0: 4000, 1: 4000})
```

- **Using SMOTE to balance the dataset**

```
num_ct = ColumnTransformer( [("scaler" , StandardScaler() ,
                              ["age", "balance", "day", "campaign", "pdays", "previous","duration"])] )
cat_ct = ColumnTransformer( [("enco" , OneHotEncoder() ,
                              ["job", "education", "marital", "default", "housing", "loan", "contact", "month",
                           "poutcome"])] )
pipe1 = Pipeline([
    ("num_pipe" , num_ct)
])

pipe2 = Pipeline([
    ("cat_pipe" , cat_ct)
])

preprocess_pipeline = FeatureUnion(transformer_list=[
        ("numerical_pipeline", pipe1),
        ("categorical_pipeline", pipe2),
    ])
```

- **Data pre-processing**

```
x_train  , x_test , y_train , y_test = train_test_split(a ,b , test_size = 0.2 , random_state = 0)
```

- **Splitting the data set in train and test.**

```
from sklearn.model_selection import cross_val_score

 # Extra Trees Classifier
extra_clf = ExtraTreesClassifier()
extra_scores = cross_val_score(extra_clf, x_train, y_train, cv=5)
extra_mean = extra_scores.mean()

# Gradient Boosting Classifier
grad_clf = GradientBoostingClassifier()
grad_scores = cross_val_score(grad_clf, x_train, y_train, cv=5)
grad_mean = grad_scores.mean()

# Random Forest Classifier
rand_clf = RandomForestClassifier()
rand_scores = cross_val_score(rand_clf, x_train, y_train, cv=5)
rand_mean = rand_scores.mean()

# LGBM Classifier
lgbm_clf = LGBMClassifier()
lgbm_scores = cross_val_score(lgbm_clf, x_train, y_train, cv=5)
lgbm_mean = lgbm_scores.mean()

d = {'Classifiers': ["ExtraTreesclf" , "GradientBosstingclf" , "RandomForest" ,"LGBMclf" ],
    'Crossval Mean Scores': [extra_mean , grad_mean , rand_mean , lgbm_mean]}

result_df = pd.DataFrame(data=d)
```

```
result_df
```

	Classifiers	Crossval Mean Scores
0	ExtraTreesclf	0.965313
1	GradientBosstingclf	0.930937
2	RandomForest	0.952344
3	LGBMclf	0.943125

- **Fitting various classification models for the dataset**

```
from sklearn.svm import SVC
svc = SVC()
svc.fit(x_train , y_train)
svc.score(x_test , y_test)
```

```
0.9175
```

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(x_train , y_train)
knn.score(x_test , y_test)
```

```
0.90625
```

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier()
mlp.fit(x_train , y_train)
mlp.score(x_test , y_test)
```

```
C:\Users\sejal Jadev\anaconda3\lib\site-packages\sklearn\neural_network\_

Stochastic Optimizer: Maximum iterations (200) reached and the optimizat
```

```
0.954375
```

```
from sklearn.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier()
gbc.fit(x_train , y_train)
gbc.score(x_test , y_test)
```

```
0.9325
```

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(x_train , y_train)
rfc.score(x_test , y_test)
```

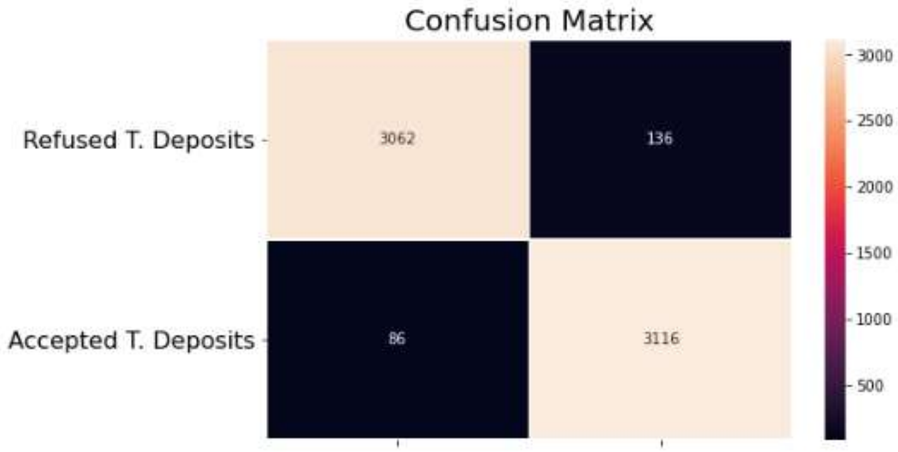
```
0.955625
```

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train , y_train)
gnb.score(x_test , y_test)
```

```
0.739375
```

- **Some of the best models. Selecting ExtraTreeClassifier for further analysis.**

```
conf_matrix = confusion_matrix(y_train, y_train_pred)
f, ax = plt.subplots(figsize=(8, 5))
sns.heatmap(conf_matrix, annot=True, fmt="d", linewidths=.5, ax=ax)
plt.title("Confusion Matrix", fontsize=20)
ax.set_xticklabels("")
ax.set_yticklabels(['Refused T. Deposits', 'Accepted T. Deposits'], fontsize=16, rotation=360)
plt.show()
```



- **Confusion Matrix output for ExtraTreeClassifier model.**

```
from sklearn.metrics import f1_score , precision_score , recall_score

print('Precision Score: ', precision_score(y_train, y_train_pred))
print('Recall Score: ', recall_score(y_train, y_train_pred))
```

```
Precision Score:  0.958179581795818
Recall Score:  0.9731417863835103
```

```
from sklearn.metrics import f1_score , roc_auc_score
f1_score(y_train, y_train_pred)
```

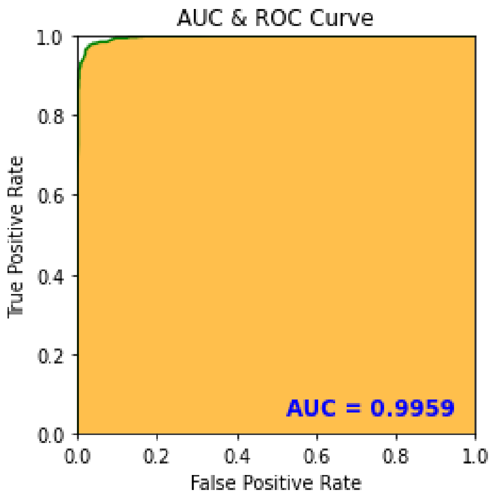
```
0.9656027269910132
```

- **Precision-Recall and f1 score**

```
y_pred = extra_clf.predict_proba(x_test)[:, 1]
y_pred.shape
```

```
(1600,)
```

```
auc = roc_auc_score(y_test, y_pred)

false_positive_rate, true_positive_rate, thresolds = roc_curve(y_test, y_pred)

plt.figure(figsize=(6, 4), dpi=70)
plt.axis('scaled')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.title("AUC & ROC Curve")
plt.plot(false_positive_rate, true_positive_rate, 'g')
plt.fill_between(false_positive_rate, true_positive_rate, facecolor='orange', alpha=0.7)
plt.text(0.95, 0.05, 'AUC = %0.4f' % auc, ha='right', fontsize=12, weight='bold', color='blue')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```



- **AUC and ROC curve giving 99% as output.**