# 0.) Import the Credit Card Fraud Data From CCLE

In [1]:
```
import pandas as pd
# from google.colab import drive
import matplotlib.pyplot as plt
import numpy as np
```

In [ ]:
```
# drive.mount('/content/gdrive/', force_remount = True)
```

```
Mounted at /content/gdrive/
```

In [2]:
```
df = pd.read_csv("fraudTest.csv")
```

In [3]:
```
df.head()
```

Out[3]:

| | Unnamed: 0 | trans_date_trans_time | cc_num | merchant | category | a |
|---|---|---|---|---|---|---|
| **0** | 0 | 2020-06-21 12:14:25 | 2291163933867244 | fraud_Kirlin and Sons | personal_care | 2 |
| **1** | 1 | 2020-06-21 12:14:33 | 3573030041201292 | fraud_Sporer-Keebler | personal_care | 29 |
| **2** | 2 | 2020-06-21 12:14:53 | 3598215285024754 | fraud_Swaniawski, Nitzsche and Welch | health_fitness | 41 |
| **3** | 3 | 2020-06-21 12:15:15 | 3591919803438423 | fraud_Haley Group | misc_pos | 60 |
| **4** | 4 | 2020-06-21 12:15:17 | 3526826139003047 | fraud_Johnston-Casper | travel | 3 |

5 rows × 23 columns

In [4]:
```
df_select = df[["trans_date_trans_time", "category", "amt", "city_p

df_select["trans_date_trans_time"] = pd.to_datetime(df_select["tran
df_select["time_var"] = [i.second for i in df_select["trans_date_tr

X = pd.get_dummies(df_select, ["category"]).drop(["trans_date_trans
y = df["is_fraud"]
```

```
/var/folders/ss/vfplm3gs0zq10q6mnjlcj0ym0000gn/T/ipykernel_2514/22
82180580.py:3: 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/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-
a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/in
dexing.html#returning-a-view-versus-a-copy)
  df_select["trans_date_trans_time"] = pd.to_datetime(df_select["t
rans_date_trans_time"])
/var/folders/ss/vfplm3gs0zq10q6mnjlcj0ym0000gn/T/ipykernel_2514/22
82180580.py:4: 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/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-
a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/in
dexing.html#returning-a-view-versus-a-copy)
  df_select["time_var"] = [i.second for i in df_select["trans_date
_trans_time"]]
```

# 1.) Use scikit learn preprocessing to split the data into 70/30 in out of sample

In [5]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [6]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
```

In [7]:
```
X_test, X_holdout, y_test, y_holdout = train_test_split(X_test, y_t
```

In [8]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_holdout = scaler.transform(X_holdout)
```

# 2.) Make three sets of training data (Oversample, Undersample and SMOTE)

In [9]:
```
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
```

In [10]:
```
ros = RandomOverSampler()
over_X, over_y = ros.fit_resample(X_train, y_train)

rus = RandomUnderSampler()
under_X, under_y = rus.fit_resample(X_train, y_train)

smote = SMOTE()
smote_X, smote_y = smote.fit_resample(X_train, y_train)
```

# 3.) Train three logistic regression models

In [11]:
```
from sklearn.linear_model import LogisticRegression
```

In [12]:
```
over_log = LogisticRegression().fit(over_X, over_y)

under_log = LogisticRegression().fit(under_X, under_y)

smote_log = LogisticRegression().fit(smote_X, smote_y)
```

# 4.) Test the three models

In [13]:
```
over_log.score(X_test, y_test)
```

Out[13]:
```
0.9082751505554356
```

In [14]:
```
under_log.score(X_test, y_test)
```

Out[14]:
```
0.8703183857578157
```

In [15]:
```
smote_log.score(X_test, y_test)
```

Out[15]:
```
0.9054199956812783
```

In [ ]:
```
# We see SMOTE performing with higher accuracy but is ACCURACY real
```

# 5.) Which performed best in Out of Sample metrics?

In [ ]:
```
# Sensitivity here in credit fraud is more important as seen from l
```

In [16]:
```
from sklearn.metrics import confusion_matrix
```

In [17]:
```
y_true = y_test
```

In [18]:
```
y_pred = over_log.predict(X_test)
cm = confusion_matrix(y_true, y_pred)
cm
```

Out[18]:
```
array([[75450,  7568],
       [   78,   262]])
```

In [19]:
```
print("Over Sample Sensitivity : ", cm[1,1] /( cm[1,0] + cm[1,1]))
```

```
Over Sample Sensitivity :  0.7705882352941177
```

In [20]:
```
y_pred = under_log.predict(X_test)
cm = confusion_matrix(y_true, y_pred)
cm
```

Out[20]:
```
array([[72283, 10735],
       [   75,   265]])
```

In [21]:
```
print("Under Sample Sensitivity : ", cm[1,1] /( cm[1,0] + cm[1,1]))
```

```
Under Sample Sensitivity :  0.7794117647058824
```

In [22]:
```
y_pred = smote_log.predict(X_test)
cm = confusion_matrix(y_true, y_pred)
cm
```

Out[22]:
```
array([[75212,  7806],
       [   78,   262]])
```

In [23]:
```
print("SMOTE Sample Sensitivity : ", cm[1,1] /( cm[1,0] + cm[1,1]))
```

```
SMOTE Sample Sensitivity :  0.7705882352941177
```

# 6.) Pick two features and plot the two classes before and after SMOTE.

In [ ]:
```
raw_temp = pd.concat([X_train, y_train], axis =1)
```

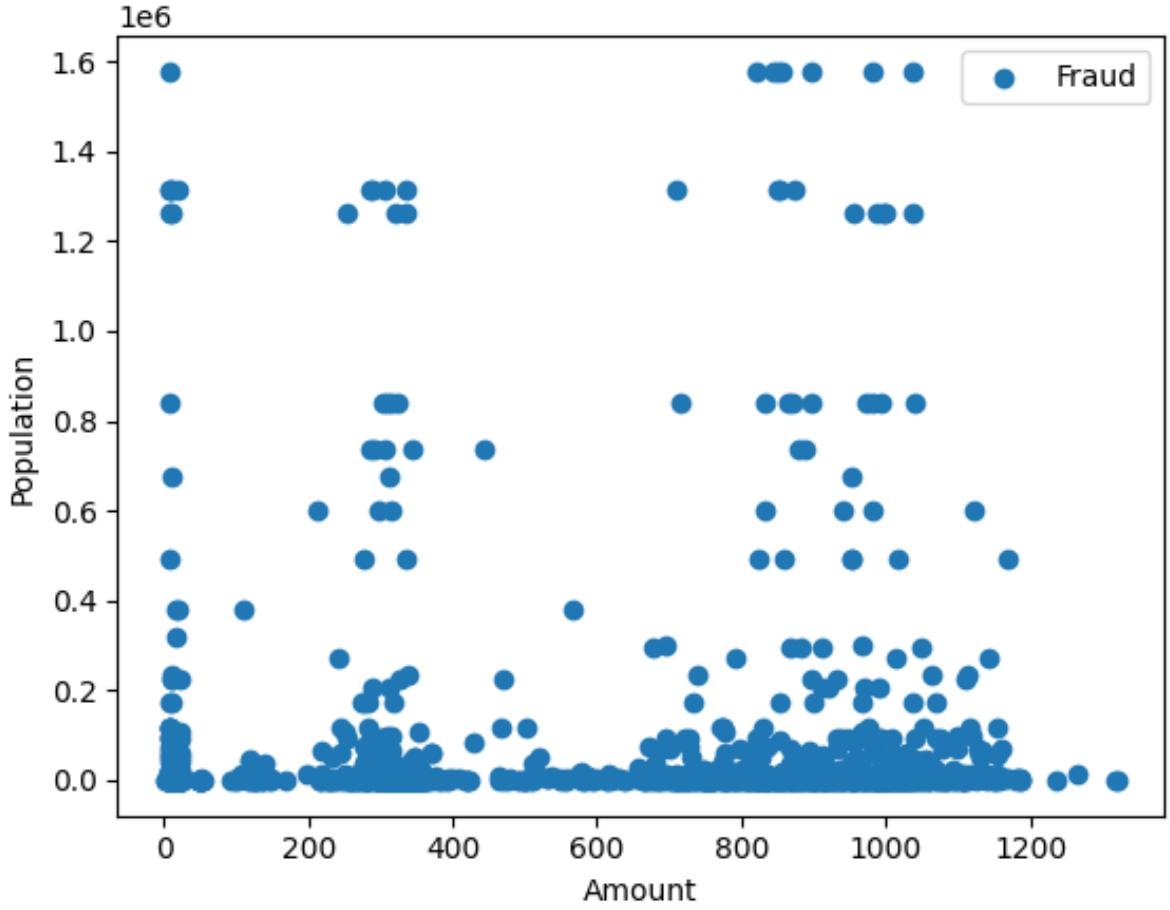
In [ ]:
```

```

In [ ]:
```
#plt.scatter(raw_temp[raw_temp["is_fraud"] == 0]["amt"], raw_temp[r

plt.scatter(raw_temp[raw_temp["is_fraud"] == 1]["amt"], raw_temp[ra
plt.legend(["Fraud", "Not Fraud"])
plt.xlabel("Amount")
plt.ylabel("Population")

plt.show()
```



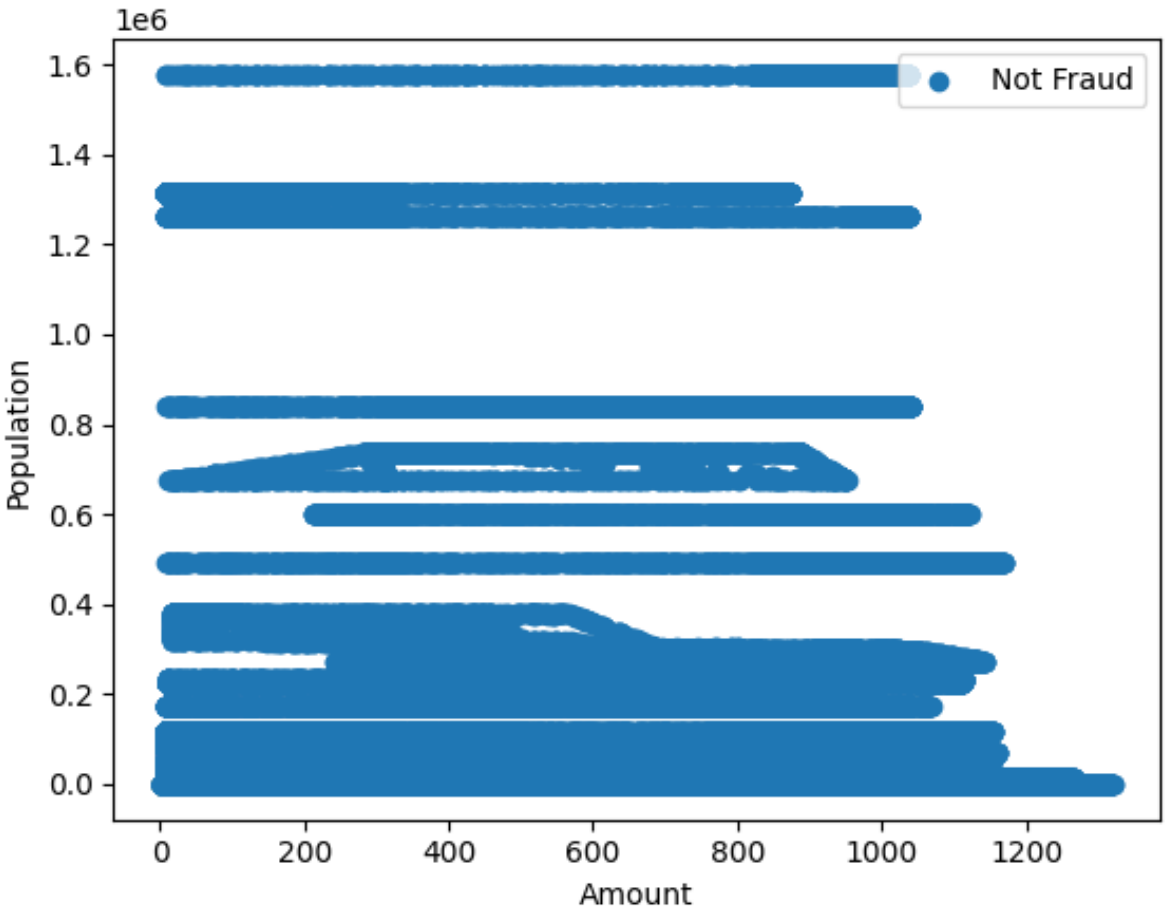

In [ ]:
```
raw_temp = pd.concat([smote_X, smote_y], axis =1)
```

In [ ]:
```
#plt.scatter(raw_temp[raw_temp["is_fraud"] == 0]["amt"], raw_temp[r

plt.scatter(raw_temp[raw_temp["is_fraud"] == 1]["amt"], raw_temp[ra
plt.legend([ "Not Fraud", "Fraud"])
plt.xlabel("Amount")
plt.ylabel("Population")

plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/IPython/core/pylabtools.py
:151: UserWarning: Creating legend with loc="best" can be slow wit
h large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```




In [ ]:

## 7.) We want to compare oversampling, Undersampling and SMOTE across our 3 models (Logistic Regression, Logistic Regression Lasso and Decision Trees).

## Make a dataframe that has a dual index and 9 Rows.

## Calculate: Sensitivity, Specificity, Precision, Recall and F1 score. for out of sample data.

## Notice any patterns across perfomance for this model. Does one totally out perform the others IE. over/under/smote or does a model perform better DT, Lasso, LR?

## Choose what you think is the best model and why. test on Holdout

In [24]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, precision_score, reca
import pandas as pd
```

In [25]:
```
resampling_methods = {
    "over": RandomOverSampler(),
    'under': RandomUnderSampler(),
    "smote": SMOTE()
}

model_configs = {
    'LOG': LogisticRegression(),
    'LASSO': LogisticRegression(penalty = 'l1', C = 2., solver = 'l
    'DecisionTree': DecisionTreeClassifier()
}
```

In [26]:
```
def calc_perf_metrics(y_true, y_pred):
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()

    sensitiivity = tp / (tp + fn)
    specificity = tn / (tp + fn)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    return(sensitiivity, specificity, precision, recall, f1)
```

In [27]:
```
trained_models = {}
results = []
```

In [28]:
```
for resample_key, resampler in resampling_methods.items():
    resample_X, resample_y = resampler.fit_resample(X_train,y_train

    for model_key, model in model_configs.items():
        combined_key = f'{resample_key}_{model_key}'

        m = model.fit(resample_X, resample_y)

        trained_models[combined_key] = m

        y_pred = m.predict(X_test)

        sensitiivity, specificity, precision, recall, f1 = calc_per

        results.append({"Model" : combined_key,
                        "Sensitivity": sensitiivity,
                        "Specificity": specificity,
                        "Precision": precision,
                        "Recall": recall,
                        "F1": f1})
```

In [29]:
```
result_df = pd.DataFrame(results)
result_df
```

Out[29]:

| | Model | Sensitivity | Specificity | Precision | Recall | F1 |
|---|---|---|---|---|---|---|
| **0** | over_LOG | 0.770588 | 221.932353 | 0.033491 | 0.770588 | 0.064192 |
| **1** | over_LASSO | 0.770588 | 221.941176 | 0.033504 | 0.770588 | 0.064216 |
| **2** | over_DecisionTree | 0.514706 | 243.750000 | 0.550314 | 0.514706 | 0.531915 |
| **3** | under_LOG | 0.770588 | 217.467647 | 0.028048 | 0.770588 | 0.054127 |
| **4** | under_LASSO | 0.773529 | 217.326471 | 0.028009 | 0.773529 | 0.054060 |
| **5** | under_DecisionTree | 0.955882 | 230.891176 | 0.067149 | 0.955882 | 0.125483 |
| **6** | smote_LOG | 0.773529 | 221.358824 | 0.032797 | 0.773529 | 0.062926 |
| **7** | smote_LASSO | 0.773529 | 221.361765 | 0.032801 | 0.773529 | 0.062934 |
| **8** | smote_DecisionTree | 0.679412 | 242.308824 | 0.267361 | 0.679412 | 0.383721 |