0.) Import the Credit Card Fraud Data From CCLE

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
#drive.mount('/content/gdrive/', force_remount = True)
```

In [4]:
```
df = pd.read_csv("fraudTest.csv")
```

In [5]:
```
df.head()
```

Out[5]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	...	lat	long	city_p
0	0	2020-06-21 12:14:25	2291163933867244	fraud_Kirlin and Sons	personal_care	2.86	Jeff	Elliott	M	351 Darlene Green	...	33.9659	-80.9355	333
1	1	2020-06-21 12:14:33	3573030041201292	fraud_Sporer-Keebler	personal_care	29.84	Joanne	Williams	F	3638 Marsh Union	...	40.3207	-110.4360	
2	2	2020-06-21 12:14:53	3598215285024754	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.28	Ashley	Lopez	F	9333 Valentine Point	...	40.6729	-73.5365	34
3	3	2020-06-21 12:15:15	3591919803438423	fraud_Haley Group	misc_pos	60.05	Brian	Williams	M	32941 Krystal Mill Apt. 552	...	28.5697	-80.8191	54
4	4	2020-06-21 12:15:17	3526826139003047	fraud_Johnston-Casper	travel	3.19	Nathan	Massey	M	5783 Evan Roads Apt. 465	...	44.2529	-85.0170	1

5 rows × 23 columns

In [6]:
```
df_select = df[["trans_date_trans_time", "category", "amt", "city_pop", "is_fraud"]]

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

X = pd.get_dummies(df_select, ["category"]).drop(["trans_date_trans_time", "is_fraud"], axis = 1)
y = df["is_fraud"]
```

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

In [7]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [8]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3)
```

In [9]:
```
X_test, X_holdout, y_test, y_holdout = train_test_split(X_test, y_test, test_size = .5)
```

In [10]:
```
X_train_df = X_train
X_test_df = X_test
X_holdout_df = X_holdout
```

In [11]:
```
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 [12]:
```
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
```

In [13]:
```
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)
```

In [14]:
```
len(y_train)
```

Out[14]: 389003

In [15]:
```
sum(y_train ==1)
```

Out[15]: 1530

In [16]:
```
sum(y_train ==0)
```

Out[16]: 387473

In [17]:
```
len(over_y)
```

Out[17]: 774946

In [18]:
```
len(under_y)
```

Out[18]: 3060

In [19]:
```
len(smote_y)
```

Out[19]: 774946

3.) Train three logistic regression models

In [20]:
```
from sklearn.linear_model import LogisticRegression
```

In [21]:
```
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 [22]:
```
over_log.score(X_test, y_test)
```

Out[22]: 0.91798027783776

In [23]:
```
under_log.score(X_test, y_test)
```

Out[23]: 0.9225029391300175

In [24]:
```
smote_log.score(X_test, y_test)
```

Out[24]: 0.9176323808152786

In [25]:
```
# We see SMOTE performing with higher accuracy but is ACCURACY really the best measure?
```

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

In [26]:
```
# Sensitivity here in credit fraud is more important as seen from last class
```

In [27]:
```
from sklearn.metrics import confusion_matrix
```

In [28]:
```
y_true = y_test
```

In [29]:
```
y_pred = over_log.predict(X_test)
cm = confusion_matrix(y_true, y_pred)
cm
```

Out[29]:
```
array([[76292,  6750],
       [   87,   229]], dtype=int64)
```

In [30]:
```python
print("Over Sample Sensitivity : ", cm[1,1] /( cm[1,0] + cm[1,1]))
```

Over Sample Sensitivity : 0.7246835443037974

In [31]:
```python
y_pred = under_log.predict(X_test)
cm = confusion_matrix(y_true, y_pred)
cm
```

Out[31]:
```
array([[76670,  6372],
       [   88,   228]], dtype=int64)
```

In [32]:
```python
print("Under Sample Sensitivity : ", cm[1,1] /( cm[1,0] + cm[1,1]))
```

Under Sample Sensitivity : 0.7215189873417721

In [33]:
```python
y_pred = smote_log.predict(X_test)
cm = confusion_matrix(y_true, y_pred)
cm
```

Out[33]:
```
array([[76263,  6779],
       [   87,   229]], dtype=int64)
```

In [34]:
```python
print("SMOTE Sample Sensitivity : ", cm[1,1] /( cm[1,0] + cm[1,1]))
```

SMOTE Sample Sensitivity : 0.7246835443037974

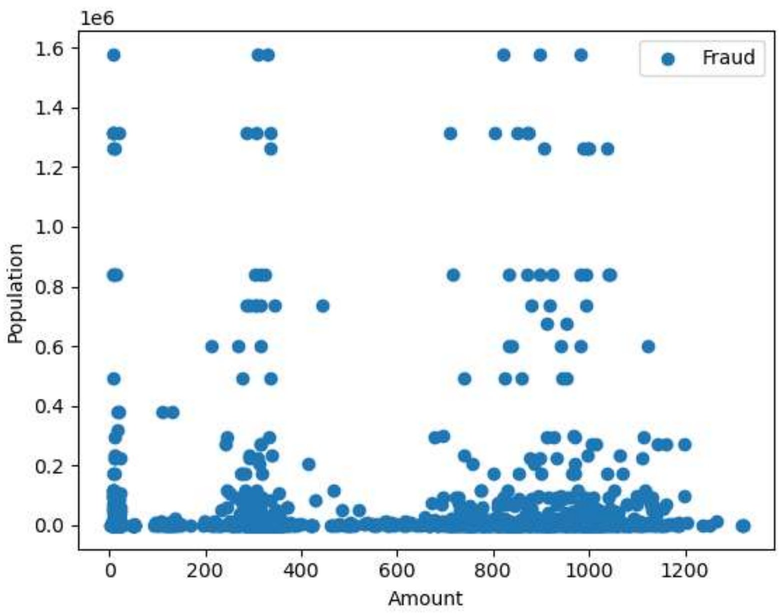
6.) Pick two features and plot the two classes before and after SMOTE.

In [35]:
```python
raw_temp = pd.concat([X_train_df, y_train], axis =1)
```

In [36]:
```python
#plt.scatter(raw_temp[raw_temp["is_fraud"] == 0]["amt"], raw_temp[raw_temp["is_fraud"] == 0]["city_pop"])

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

plt.show()
```

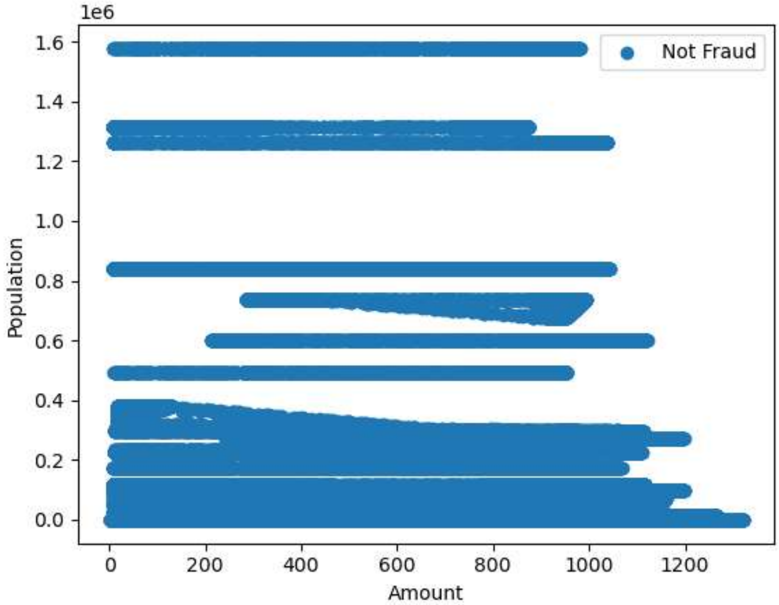


In [37]:
```python
smote_X_df, smote_y = smote.fit_resample(X_train_df,y_train)
raw_temp = pd.concat([smote_X_df, smote_y], axis =1)
```

In [38]:
```
#plt.scatter(raw_temp[raw_temp["is_fraud"] == 0]["amt"], raw_temp[raw_temp["is_fraud"] == 0]["city_pop"])

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

plt.show()
```



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 [39]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score
import pandas as pd
```

In [40]:
```
resampling_methods = {
    "over": RandomOverSampler(),
    "under":RandomUnderSampler(),
    "smote": SMOTE()
}

model_configs ={
    "LOG": LogisticRegression(),
    "LASSO": LogisticRegression(penalty = "l1", solver = "liblinear", C = .5),
    "DecisionTree": DecisionTreeClassifier()
}
```

In [41]:
```
trained_models = {}
results = []
```

In [42]:
```
def calc_perf_metric(y_true, y_pred):
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()

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

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

In [43]:
```
for resample_key, resampler in resampling_methods.items():
    resample_X, resample_y = resampler.fit_resample(X_train, y_train)

    for model_name, model in model_configs.items():

        combined_key = f"{resample_key}_{model_name}"
        m = model.fit(resample_X, resample_y)
        trained_models[combined_key] = m
        y_pred=m.predict(X_test)
        sensitivity, specificity, precision, recall, f1=calc_perf_metric(y_test, y_pred)
        results.append({"Model":combined_key,
                       "Sensitivity":sensitivity,
                       "Specificity":specificity,
                       "Precision":precision,
                       "Recall":recall,
                       "F1":f1})
```

In [44]:
```
results_df = pd.DataFrame(results)
results_df
```

Out[44]:

	Model	Sensitivity	Specificity	Precision	Recall	F1
0	over_LOG	0.724684	0.917825	0.032468	0.724684	0.062152
1	over_LASSO	0.724684	0.917849	0.032478	0.724684	0.062169
2	over_DecisionTree	0.563291	0.998531	0.593333	0.563291	0.577922
3	under_LOG	0.702532	0.935960	0.040072	0.702532	0.075820
4	under_LASSO	0.702532	0.935996	0.040094	0.702532	0.075859
5	under_DecisionTree	0.955696	0.943547	0.060521	0.955696	0.113833
6	smote_LOG	0.724684	0.917283	0.032263	0.724684	0.061775
7	smote_LASSO	0.724684	0.917295	0.032267	0.724684	0.061783
8	smote_DecisionTree	0.718354	0.993618	0.299868	0.718354	0.423113