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 [2]:
```
#drive.mount('/content/gdrive/', force_remount = True)
```

In [3]:
```
df = pd.read_csv("fraudTest.csv")
```

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

Out[4]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first
0	0	2020-06-21 12:14:25	2291163933867244	fraud_Kirlin and Sons	personal_care	2.86	Jeff
1	1	2020-06-21 12:14:33	3573030041201292	fraud_Sporer-Keebler	personal_care	29.84	Joanne
2	2	2020-06-21 12:14:53	3598215285024754	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.28	Ashley
3	3	2020-06-21 12:15:15	3591919803438423	fraud_Haley Group	misc_pos	60.05	Brian
4	4	2020-06-21 12:15:17	3526826139003047	fraud_Johnston-Casper	travel	3.19	Nathan

5 rows × 23 columns

In [5]:
```
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"]
y = df["is_fraud"]
```

```
C:\Users\antek\AppData\Local\Temp\ipykernel_13944\2282180580.py:3: SettingWithCopyWar
ning:
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-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df_select["trans_date_trans_time"] = pd.to_datetime(df_select["trans_date_trans_tim
e"])
C:\Users\antek\AppData\Local\Temp\ipykernel_13944\2282180580.py:4: SettingWithCopyWar
ning:
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-docs/stable/us
er_guide/indexing.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 [6]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
```

In [7]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3)
```

In [8]:
```
X_test, X_holdout, y_test, y_holdout = train_test_split(X_test, y_test, test_size = .5
```

In [9]:
```
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 [10]:
```
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
```

In [11]:
```
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 [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

3.) Train three logistic regression models

In [12]:
```
from sklearn.linear_model import LogisticRegression
```

In [13]:
```
over_log = LogisticRegression().fit(over_X, over_y)

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

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

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

4.) Test the three models

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

Out[14]:
```
0.9199237025840351
```

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

Out[15]:
```
0.8988819309484393
```

In [16]:
```
smote_log.score(X_test, y_test)
```

Out[16]:
```
0.9190239689052041
```

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

In [ ]:
```
```

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

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

In [17]:
```
from sklearn.metrics import confusion_matrix
```

In [18]:
```
y_true = y_test
```

In [19]:
```
y_pred = over_log.predict(X_test)
cm = confusion_matrix(y_true, y_pred)
cm
```

Out[19]:
```
array([[76455,  6601],
       [   74,   228]], dtype=int64)
```

In [20]:
```
print("Over Sample Sensitivity : ", cm[1,1] /( cm[1,0] + cm[1,1]))
```

```
Over Sample Sensitivity :  0.7549668874172185
```

In [21]:
```
y_pred = under_log.predict(X_test)
cm = confusion_matrix(y_true, y_pred)
cm
```

Out[21]:
```
array([[74701,  8355],
       [   74,   228]], dtype=int64)
```

In [22]:
```
print("Under Sample Sensitivity : ", cm[1,1] /( cm[1,0] + cm[1,1]))
```

```
Under Sample Sensitivity :  0.7549668874172185
```

In [23]:
```
y_pred = smote_log.predict(X_test)
cm = confusion_matrix(y_true, y_pred)
cm
```

Out[23]:
```
array([[76380,  6676],
       [   74,   228]], dtype=int64)
```

In [24]:
```
print("SMOTE Sample Sensitivity : ", cm[1,1] /( cm[1,0] + cm[1,1]))
```

```
SMOTE Sample Sensitivity :  0.7549668874172185
```

In [ ]:
```
```

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

In [ ]:
```
```

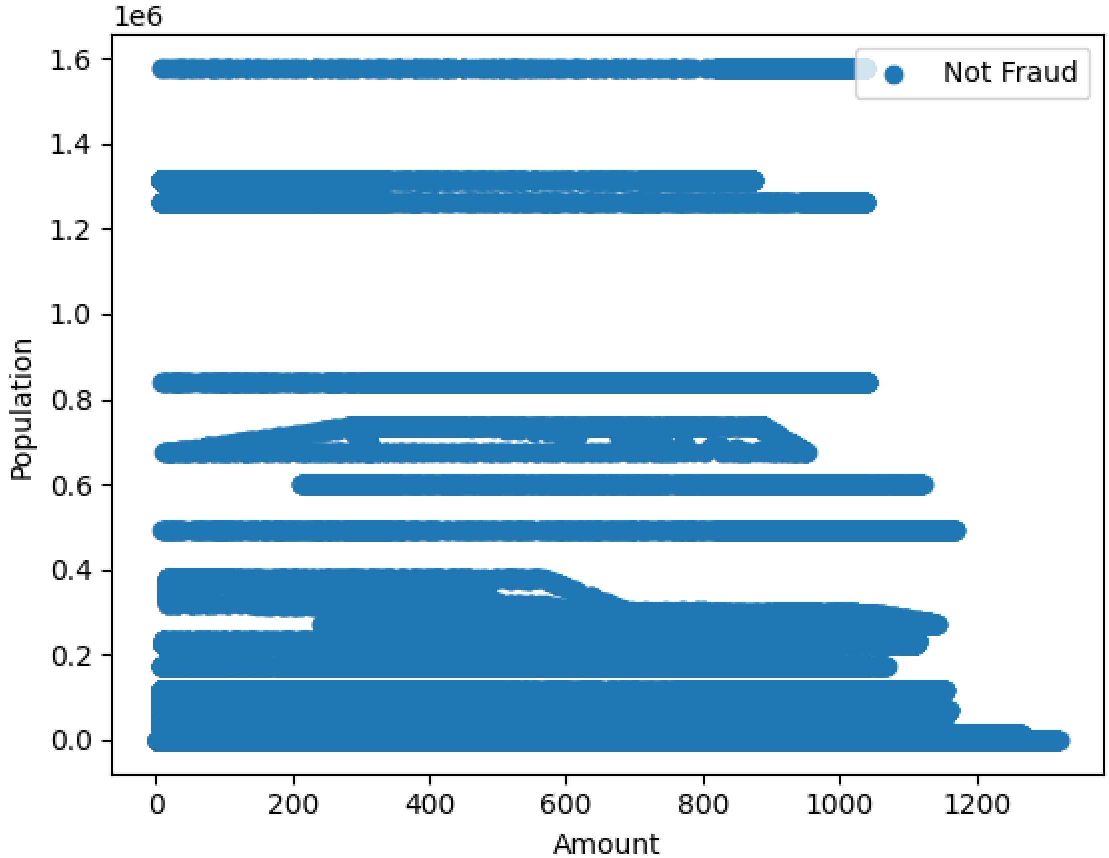
In [ ]:
```
raw_temp = pd.concat([smote_X, smote_y], axis =1)
```

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

plt.scatter(raw_temp[raw_temp["is_fraud"] == 1]["amt"], raw_temp[raw_temp["is_fraud"]
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 with 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 [33]:
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score
import pandas as pd
```

In [34]:
```python
resampling_methods = {
    'over' : RandomOverSampler(),
    'under' : RandomUnderSampler(),
    'smote' : SMOTE()
}
model_configs = {
    'LOG' : LogisticRegression(),
    'Lasso' : LogisticRegression(penalty = 'l1', C = 2,solver = 'liblinear'),
    'DTREE' : DecisionTreeClassifier()
}
```

In [53]:
```python
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 [54]:
```python
trained_models = {}
results = []
```

In [ ]:
```python

```

In [55]:
```python
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}"
        model.fit(resample_X,resample_y)
        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,
                         'Precision' : precision,
                        'Recall' : recall,
                        'f1' : f1
                       })
```

In [57]:

```
results_df = pd.DataFrame(results)
```

In [59]:

```
results_df
```

Out[59]:

	Model	Sensitivity	Precision	Recall	f1
0	over_LOG	0.754967	0.032469	0.754967	0.062261
1	over_Lasso	0.754967	0.032488	0.754967	0.062295
2	over_DTREE	0.562914	0.543131	0.562914	0.552846
3	under_LOG	0.754967	0.027829	0.754967	0.053679
4	under_Lasso	0.754967	0.027560	0.754967	0.053178
5	under_DTREE	0.960265	0.056486	0.960265	0.106696
6	smote_LOG	0.754967	0.031250	0.754967	0.060016
7	smote_Lasso	0.754967	0.031267	0.754967	0.060047
8	smote_DTREE	0.695364	0.248815	0.695364	0.366492

overall it appears that the undersampled dtree performed the best for this data set however there may still be issues due to the large amount of left out data