Kehua Chu (uid: 806153163)

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

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

Out[3]:

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

5 rows × 23 columns

In [4]:
```
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"], axi
y = df["is_fraud"]
```

```
C:\Users\Kacie\AppData\Local\Temp\ipykernel_2088\2282180580.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/pandas-docs/stable/user_guid
e/indexing.html#returning-a-view-versus-a-copy
  df_select["trans_date_trans_time"] = pd.to_datetime(df_select["trans_date_trans_time"])
C:\Users\Kacie\AppData\Local\Temp\ipykernel_2088\2282180580.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/pandas-docs/stable/user_guid
e/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 [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 = .3)
```

In [7]:
```
X_test, X_holdout, y_test, y_holdout = train_test_split(X_test, y_test, test_size = .5)
```

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)
```

```
e:\..Kacie\AnacondaKC\Lib\site-packages\joblib\externals\loky\backend\context.py:110: UserWa
rning: Could not find the number of physical cores for the following reason:
[WinError 2] 系统找不到指定的文件。
Returning the number of logical cores instead. You can silence this warning by setting LOKY_
MAX_CPU_COUNT to the number of cores you want to use.
  warnings.warn(
  File "e:\..Kacie\AnacondaKC\Lib\site-packages\joblib\externals\loky\backend\context.py", l
ine 199, in _count_physical_cores
    cpu_info = subprocess.run(
               ^^^^^^^^^^^^^^^
  File "e:\..Kacie\AnacondaKC\Lib\subprocess.py", line 548, in run
    with Popen(*popenargs, **kwargs) as process:
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "e:\..Kacie\AnacondaKC\Lib\subprocess.py", line 1026, in __init__
    self._execute_child(args, executable, preexec_fn, close_fds,
  File "e:\..Kacie\AnacondaKC\Lib\subprocess.py", line 1538, in _execute_child
    hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
                       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

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]:
```
0.9280812879387701
```

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

Out[14]:
```
0.9281412701840255
```

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

Out[15]:
```
0.9248782360421315
```

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

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

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

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([[77138,  5926],
       [   69,   225]], dtype=int64)
```

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

```
Over Sample Sensitivity :  0.7653061224489796
```

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

Out[20]:
```
array([[77142,  5922],
       [   68,   226]], dtype=int64)
```

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

```
Under Sample Sensitivity :  0.7687074829931972
```

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

Out[22]:
```
array([[76871,  6193],
       [   69,   225]], dtype=int64)
```

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

```
SMOTE Sample Sensitivity :  0.7653061224489796
```

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

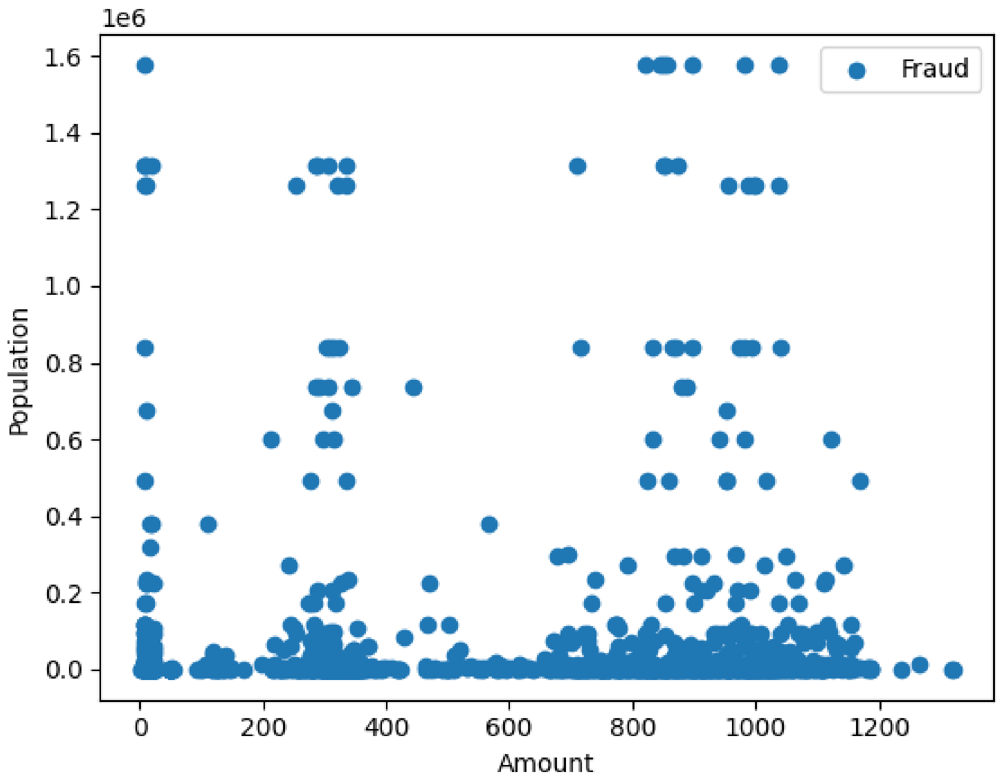
In [25]:
```
X_train_df = pd.DataFrame(X_train)
```

In [27]:
```
raw_temp = pd.concat([X_train_df, y_train], axis =1)
```

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

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

plt.show()
```



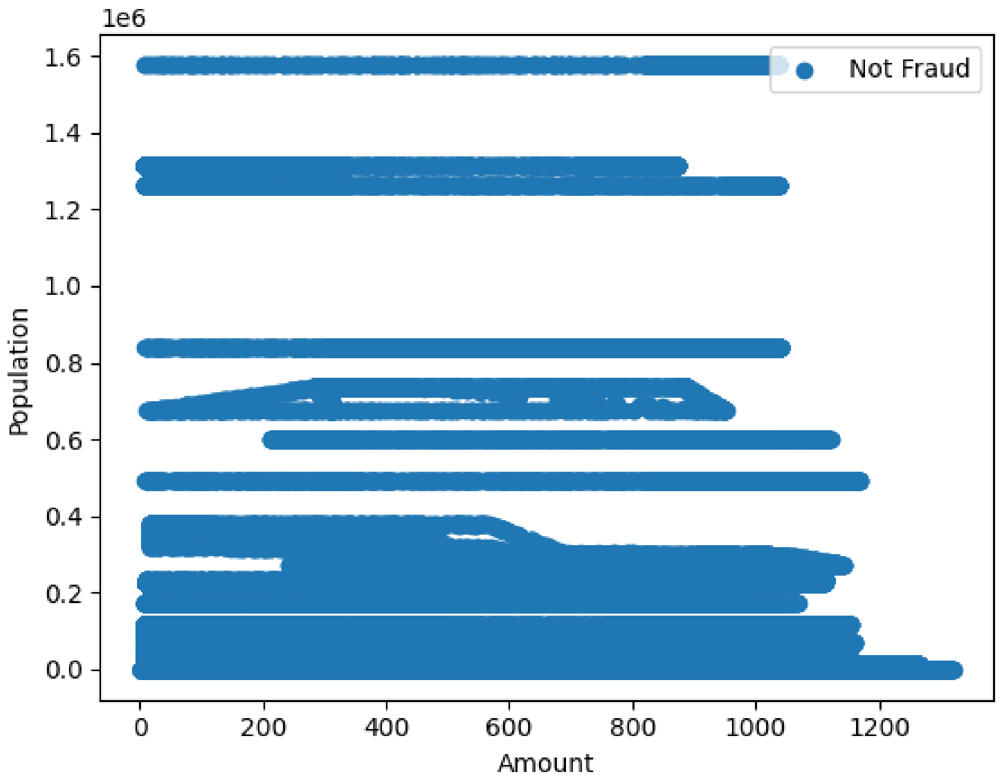
In [ ]:
```
raw_temp = pd.concat([smote_X, smote_y], axis =1)
```

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

plt.scatter(raw_temp[raw_temp["is_fraud"] == 1]["amt"], raw_temp[raw_temp["is_fraud"] == 1]
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: Creatin
g 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 [24]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score
```

```python
import pandas as pd
```

In [25]:
```python
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
```

In [26]:
```python
resampling_methods = {
    'over': RandomOverSampler(),
    'under':RandomUnderSampler(),
    'smote':SMOTE()
}
```

In [27]:
```python
model_configs = {
    'LOG': LogisticRegression(),
    'LASSO': LogisticRegression(penalty = 'l1', solver = 'liblinear', C = 0.5),
    'DecisionTree': DecisionTreeClassifier()
}
```

In [36]:
```python
trained_models = {}
results = []
```

In [30]:
```python
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 [37]:
```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}'

        m = model.fit(resample_X, resample_y)

        trained_models[combined_key] = m

        y_pred = m.predict(X_test)

        sensitiivity, specificity, precision, recall, f1 = calc_perf_metrics(y_test, y_prec

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

In [38]:
```python
result_df = pd.DataFrame(results)
```

In [39]:
```python
result_df
```

Out[39]:

	Model	Sensitivity	Specificity	Precision	Recall	F1
0	over_LOG	0.765306	261.812925	0.035624	0.765306	0.068079
1	over_LASSO	0.765306	261.816327	0.035629	0.765306	0.068089
2	over_DecisionTree	0.561224	282.078231	0.553691	0.561224	0.557432
3	under_LOG	0.765306	261.071429	0.034435	0.765306	0.065905
4	under_LASSO	0.765306	261.884354	0.035743	0.765306	0.068296
5	under_DecisionTree	0.948980	268.132653	0.061835	0.948980	0.116105
6	smote_LOG	0.765306	261.751701	0.035523	0.765306	0.067894
7	smote_LASSO	0.765306	261.748299	0.035517	0.765306	0.067884
8	smote_DecisionTree	0.734694	280.486395	0.264382	0.734694	0.388839