0.) Import the Credit Card Fraud Data From CCLE

In [3]:
```
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)
```

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

In [4]:
```
df = pd.read_csv("/content/gdrive/MyDrive/W24ML Code/Data/fraudTest.csv")
```

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

Out[5]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	s
0	0	2020-06-21 12:14:25	2291163933867244	fraud_Kirlin and Sons	personal_care	2.86	Jeff	Elliott	M	Da C
1	1	2020-06-21 12:14:33	3573030041201292	fraud_Sporer-Keebler	personal_care	29.84	Joanne	Williams	F	N U
2	2	2020-06-21 12:14:53	3598215285024754	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.28	Ashley	Lopez	F	Vale
3	3	2020-06-21 12:15:15	3591919803438423	fraud_Haley Group	misc_pos	60.05	Brian	Williams	M	3 Kı Mil
4	4	2020-06-21 12:15:17	3526826139003047	fraud_Johnston-Casper	travel	3.19	Nathan	Massey	M	F Apt

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"]
```

```
/var/folders/j8/qj6z29_s2qj2dwzv274nkt9h0000gp/T/ipykernel_20135/2282180580.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_guide/indexing.h
tml#returning-a-view-versus-a-copy
  df_select["trans_date_trans_time"] = pd.to_datetime(df_select["trans_date_trans_time"])
/var/folders/j8/qj6z29_s2qj2dwzv274nkt9h0000gp/T/ipykernel_20135/2282180580.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_guide/indexing.h
tml#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 [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]:
```
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 [13]:
```
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
```

In [14]:
```
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 [15]:
```
from sklearn.linear_model import LogisticRegression
```

In [16]:
```
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 [17]:
```
over_log.score(X_test, y_test)
```

Out[17]:
```
0.9051800667002567
```

In [18]:
```
under_log.score(X_test, y_test)
```

Out[18]:
```
0.9072194630389405
```

In [19]:
```
smote_log.score(X_test, y_test)
```

Out[19]:
```
0.9042923294704768
```

In [20]:
```
# We see SMOTE performing with higher accuracy but is ACCURACY really the best measure - it is not.
```

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

In [21]:
```
# Sensitivity here in credit fraud is more important as seen from last class
```

In [22]:
```
from sklearn.metrics import confusion_matrix
```

In [23]:
```
y_true = y_test
```

In [24]:
```
y_pred = over_log.predict(X_test)
cm = confusion_matrix(y_true, y_pred)
cm
```

Out[24]:
```
array([[75226,  7830],
       [   74,   228]])
```

In [25]:
```
print("Over Sample Sensitivity : ", cm[1,1] /( cm[1,0] + cm[1,1]))
```

```
Over Sample Sensitivity :  0.7549668874172185
```

In [26]:
```
y_pred = under_log.predict(X_test)
cm = confusion_matrix(y_true, y_pred)
cm
```

Out[26]:
```
array([[75397,  7659],
       [   75,   227]])
```

In [27]:
```
print("Under Sample Sensitivity : ", cm[1,1] /( cm[1,0] + cm[1,1]))
```

```
Under Sample Sensitivity :  0.7516556291390728
```

In [28]:
```
y_pred = smote_log.predict(X_test)
cm = confusion_matrix(y_true, y_pred)
cm
```

Out[28]:
```
array([[75152,  7904],
       [   74,   228]])
```

In [29]:
```
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.

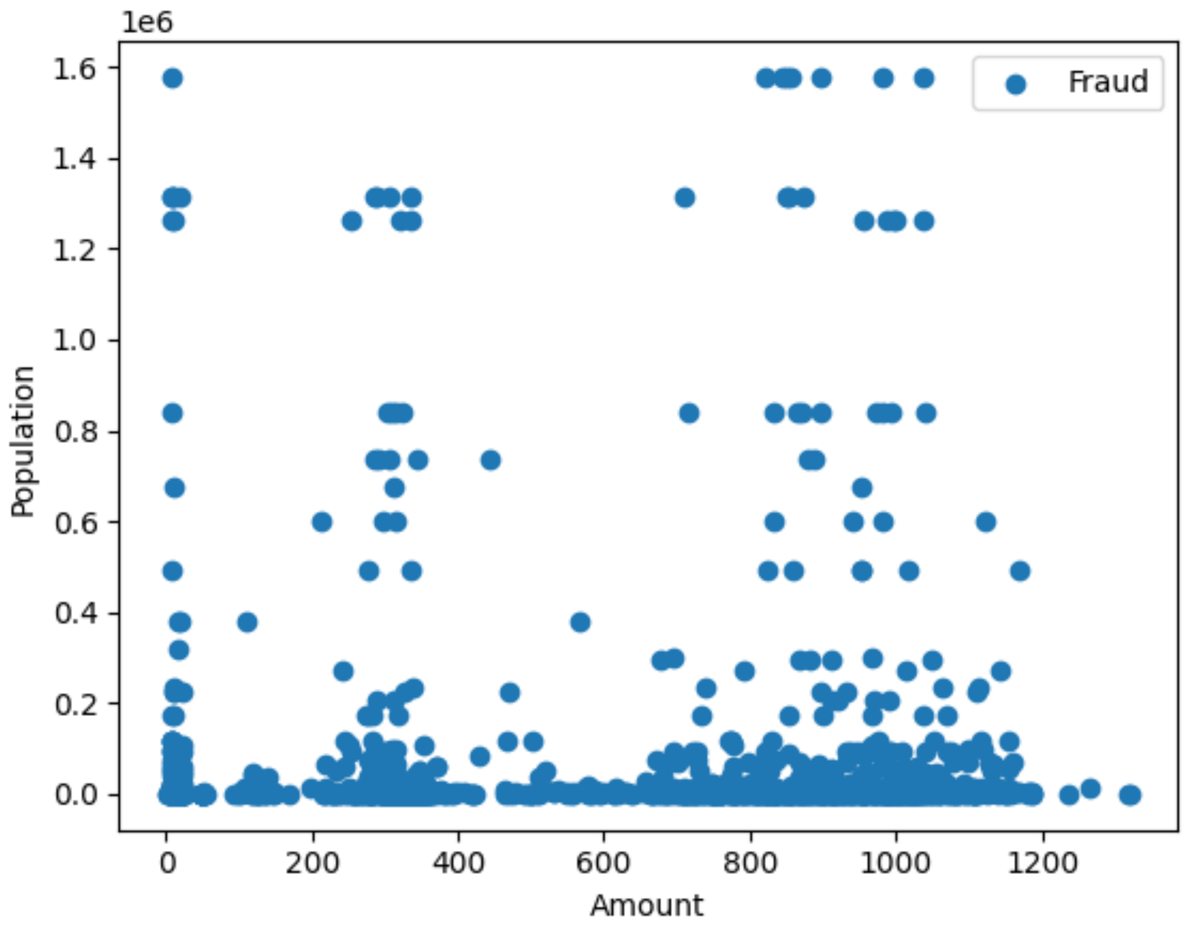
In [ ]:
```
raw_temp = pd.concat([X_train, y_train], axis =1)
```

In [ ]:

In [ ]:
```
#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()
```



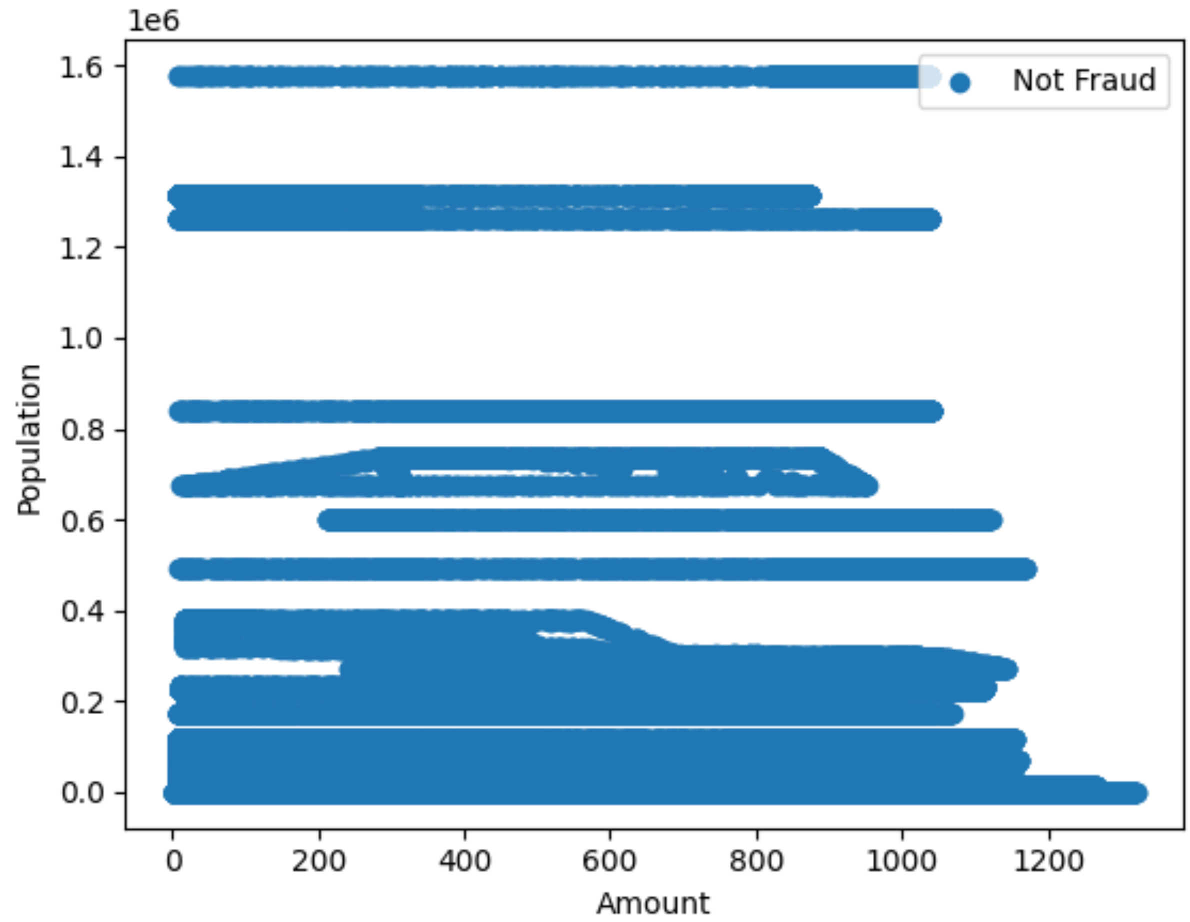
In [ ]:
```
raw_temp = pd.concat([smote_X, smote_y], axis =1)
```

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

```
/usr/local/lib/python3.10/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Creating legend wit
h 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 [31]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score
import pandas as pd
```

In [33]:
```
resampling_methods = {
    "over": RandomOverSampler(),
    "under":RandomUnderSampler(),
    "smote":SMOTE()
}

model_configs = {
    "LOG":LogisticRegression(),
    "LASSO": LogisticRegression(penalty = "l1", C = 2., solver = "liblinear"),
    "DTREE":DecisionTreeClassifier()


}
```

In [35]:
```
trained_models= {}
```

In [45]:
```
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}"
        trained_models[combined_key] = model.fit(resample_X, resample_y)
```

In [46]:
```
trained_models
```

Out[46]:
```
{'over_LOG': LogisticRegression(),
 'over_LASSO': LogisticRegression(C=2.0, penalty='l1', solver='liblinear'),
 'over_DTREE': DecisionTreeClassifier(),
 'under_LOG': LogisticRegression(),
 'under_LASSO': LogisticRegression(C=2.0, penalty='l1', solver='liblinear'),
 'under_DTREE': DecisionTreeClassifier(),
 'smote_LOG': LogisticRegression(),
 'smote_LASSO': LogisticRegression(C=2.0, penalty='l1', solver='liblinear'),
 'smote_DTREE': DecisionTreeClassifier()}
```

In [66]:
```
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score
import pandas as pd

# Initialize an empty list to store results
def calc_perf_metric(y_true, y_pred):
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()

    sensitivity = tp / (tp + fn)  # Corrected formula
    specificity = tn / (tn + fp)  # Corrected formula
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)  # Recall is the same as sensitivity
    f1 = f1_score(y_true, y_pred)

    # Return a dictionary of the calculated metrics
    return(sensitivity,specificity,precision,recall,f1)
```

In [67]:
```
trained_models = {}
results = []
```

In [68]:
```
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)
        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_score': f1


                       })

        #results.append(calc_perf_metric(y_true, y_pred))
```

In [69]:
```
results_df = pd.DataFrame(results)
```

In [70]:
```
results_df
```

Out[70]:

	Model	sensitivity	specificity	precision	recall	f1_score
0	over_LOG	0.751656	0.906750	0.028475	0.751656	0.054871
1	over_LASSO	0.751656	0.906786	0.028485	0.751656	0.054891
2	over_DTREE	0.552980	0.998531	0.577855	0.552980	0.565144
3	under_LOG	0.758278	0.875385	0.021647	0.758278	0.042092
4	under_LASSO	0.761589	0.874795	0.021639	0.761589	0.042082
5	under_DTREE	0.943709	0.949528	0.063659	0.943709	0.119272
6	smote_LOG	0.754967	0.907376	0.028784	0.754967	0.055454
7	smote_LASSO	0.754967	0.907376	0.028784	0.754967	0.055454
8	smote_DTREE	0.725166	0.993896	0.301653	0.725166	0.426070

Performance Analysis

1. **Sensitivity (Recall)**:

   - The highest sensitivity is observed with the `under_DTREE` model, indicating it is best at identifying positive cases.
   - Both Logistic Regression models (`over_LOG` , `smote_LOG`) and the Lasso models show similar sensitivity across the resampling methods, suggesting consistency in identifying true positives regardless of the resampling strategy.
2. **Specificity**:

   - The Decision Tree models (`over_DTREE` , `smote_DTREE`) exhibit high specificity, particularly with the `over_DTREE` achieving nearly perfect specificity. This indicates strong performance in identifying true negatives.
3. **Precision**:

   - The `over_DTREE` model significantly outperforms others in precision, which suggests it has the lowest rate of false positives. This is crucial in scenarios where the cost of a false positive is high.
4. **F1 Score**:

   - The `smote_DTREE` model stands out with the highest F1 score, indicating a balanced performance between precision and recall. This balance is important for achieving overall accuracy.

Pattern Recognition

- **Decision Trees with SMOTE** (`smote_DTREE`) shows a compelling balance across all metrics, particularly in achieving a high F1 score, which suggests a balanced trade-off between precision and recall.
- **Oversampling** tends to favor specificity but at the cost of precision, as seen in the `over_DTREE` model.
- **Undersampling** significantly enhances sensitivity, especially for the Decision Tree model (`under_DTREE`), making it an excellent choice for applications where missing a positive case (false negative) is critical.
- Logistic Regression models, both standard and with Lasso, display consistency across resampling techniques but do not excel in any particular metric compared to Decision Trees.

Best Model Selection

Considering the balance between all metrics, **Decision Trees with SMOTE (`smote_DTREE`)** appears to be the best model due to its:

- High sensitivity, ensuring most positive cases are identified.
- Excellent specificity, minimizing false positives.
- Strong F1 score, indicating a balanced precision and recall, making it versatile for various applications.

This model strikes a good balance between identifying positive cases without significantly increasing the false positives, making it a robust choice for many scenarios. Testing this model on a holdout dataset would be the next step to validate its performance on unseen data and ensure its generalizability and effectiveness.

In [ ]: