Habitability of Exoplanets

Ben Prentice and Noah Psutka

Problem Statement

Data from The Habitable Exoplanets Catalog

binary/trianary classification prediction model

The class label we predicted will be in regards to the habitability of an exoplanet;

not-habitable = 0, habitable = 1, strong habitability = 2

Project Data/Model Information

S_METALLICITY	abundance of elements present in the planet that are heavier than hydrogen and helium
S_AGE	age of the planet
S_TIDAL_LOCK	situation in which an astronomical object's orbital period matches its rotational period
P_TEMP_EQUIL_MIN	minimum temperature estimated in degrees Kelvin
P_TEMP_EQUIL	average temperature estimated in degrees Kelvin
P_TPERI	time of passage at the periapse for eccentric orbits
P_TEMP_EQUIL_MAX	maximum temperature estimated in degrees Kelvin
P_HABZONE_CON	not in the habitable zone of a star (binary classification)
P_ESI	Earth Similarity Index, a measure of similarity to Earth's stellar flux, and mass or radius (Earth=1.0)
P_HABZONE_OPT	in the habitable zone of a star (binary classification)

* Typical algorithms used for binary classification are Logistic Regression, and SVM's. These will be the benchmarks used to determine the output class labels.
Imbalanced (Binary) Classification: Many of our data points for habitability belong to the non-habitable label; Therefore we will be using cost-sensitive versions of logistic regression, SVM's and Decision Trees.

Data Preprocessing

```
df.info()
df.shape
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4048 entries, 0 to 4047
Columns: 112 entries, P_NAME to P_SEMI_MAJOR_AXIS_EST
dtypes: float64(94), int64(4), object(14)
memory usage: 3.5+ MB

(4048, 112)
```

```
df.isnull().sum().sum()
```

```
149723
```

```
In [4]: df.isnull().sum().sum()
Out[4]: 149723
```

453,376 total data points

149,723 null data points

about 33.02% of our data is null values

```
In [5]: df.fillna(df.mean(),inplace=True)

        df.isnull().sum().sum()
```

```
C:\Users\Noah\AppData\Local\Temp\ipy
eductions (with 'numeric_only=None'
s before calling the reduction.
  df.fillna(df.mean(),inplace=True)
```

```
Out[5]: 31494
```

```
In [6]: df = df.dropna(axis = 1)
        df.isnull().sum().sum()
Out[6]: 0
```

```
In [7]: df.shape
Out[7]: (4048, 101)
```

Feature Extraction

```
corr_habitable = corr_matrix['P_HABITABLE']
corr_habitable = corr_habitable.dropna()
corr_habitable = corr_habitable.abs()
corr_habitable = corr_habitable.drop('P_HABITABLE')
corr_habitable = corr_habitable.sort_values()
```

```
top10 = corr_habitable.tail(10)
top25 = corr_habitable.tail(25)
```

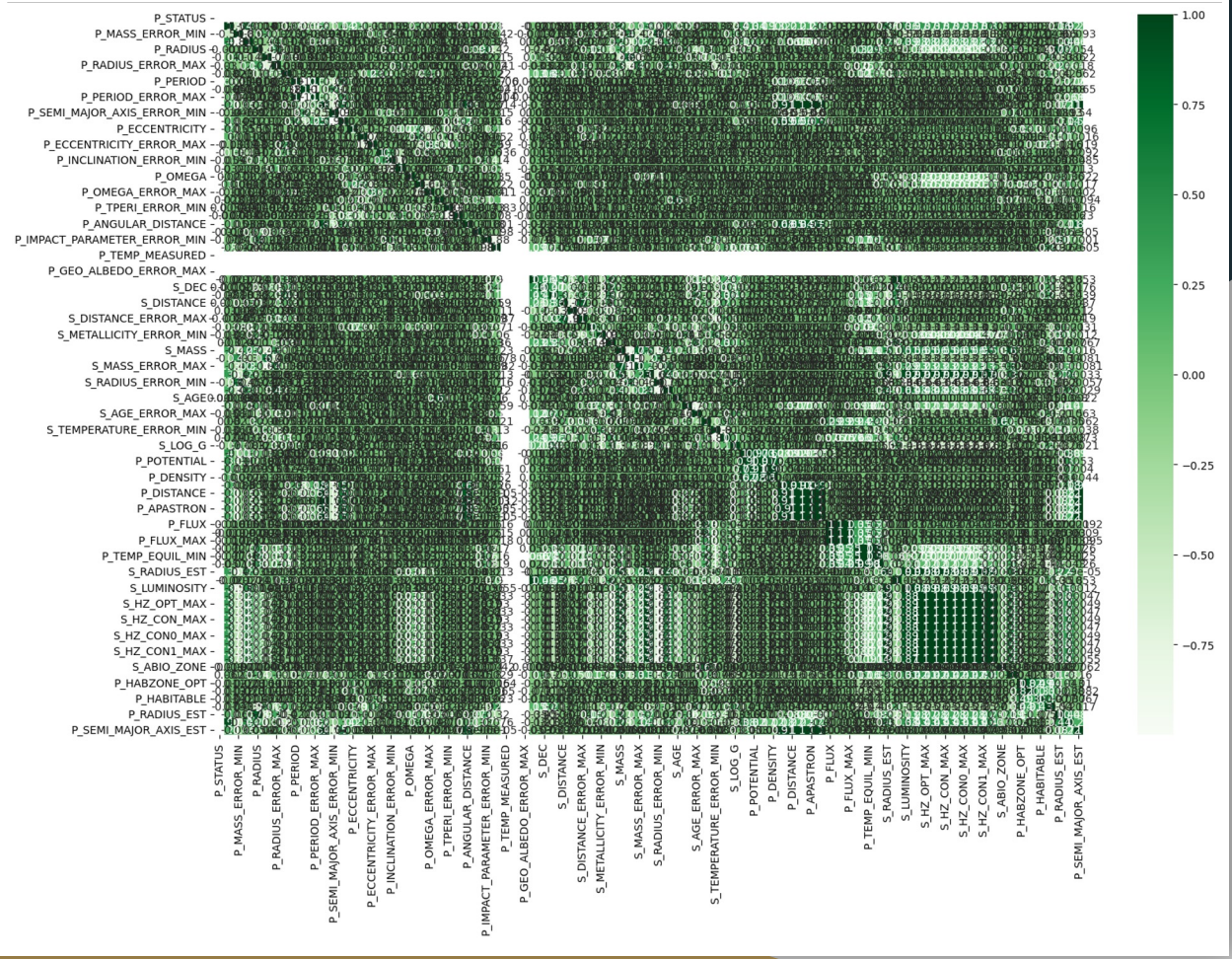
```
print(top10)
```

```
S_METALLICITY       0.092922
S_AGE               0.110499
S_TIDAL_LOCK        0.136728
P_TEMP_EQUIL_MIN    0.140296
P_TEMP_EQUIL        0.141131
P_TPERI             0.141285
P_TEMP_EQUIL_MAX    0.142674
P_HABZONE_CON       0.298875
P_ESI               0.427036
P_HABZONE_OPT       0.491481
Name: P_HABITABLE, dtype: float64
```

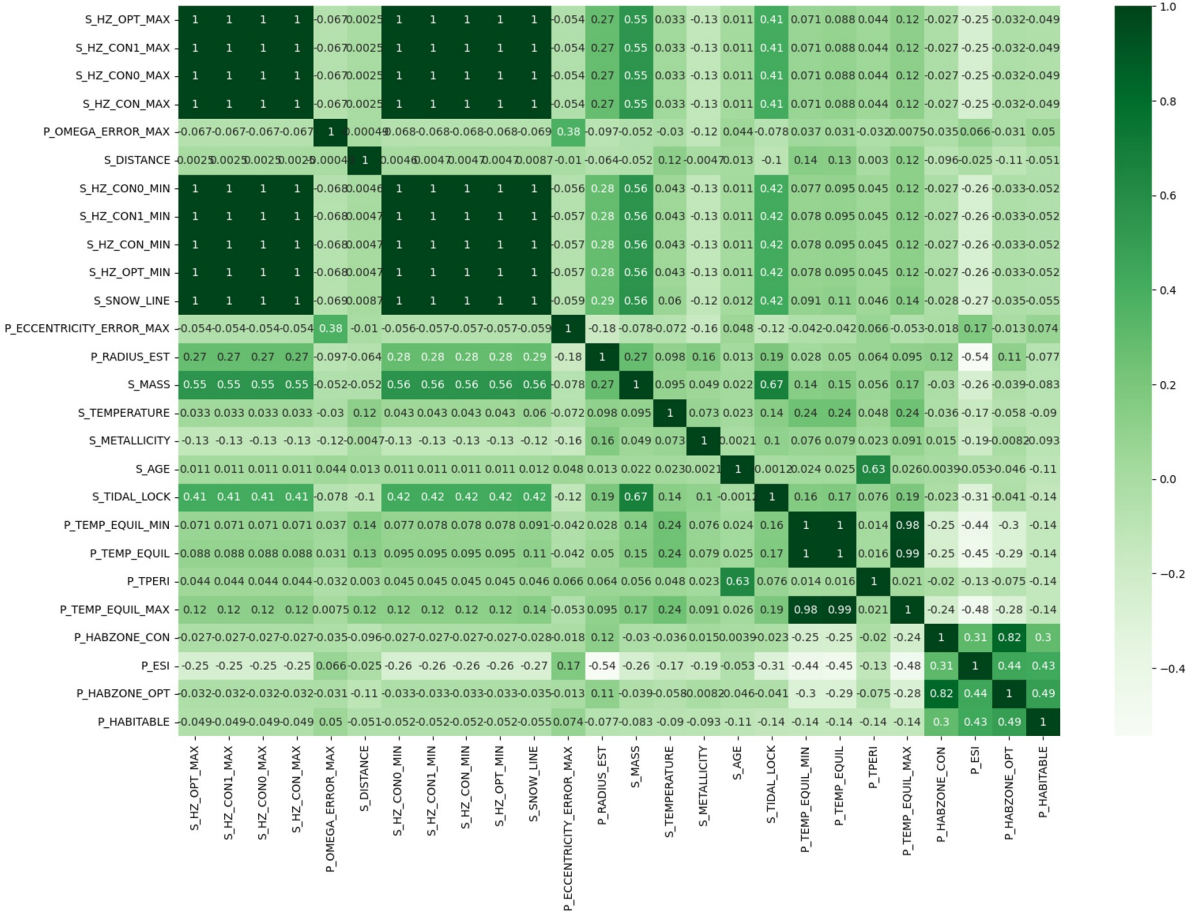
```
print(top25)
```

```
S_HZ_OPT_MAX                0.049386
S_HZ_CON1_MAX               0.049387
S_HZ_CON0_MAX               0.049387
S_HZ_CON_MAX                0.049387
P_OMEGA_ERROR_MAX           0.049910
S_DISTANCE                  0.051299
S_HZ_CON0_MIN               0.051827
S_HZ_CON1_MIN               0.051897
S_HZ_CON_MIN                0.051900
S_HZ_OPT_MIN                0.051901
S_SNOW_LINE                 0.055036
P_ECCENTRICITY_ERROR_MAX    0.074497
P_RADIUS_EST                0.076826
S_MASS                      0.083176
S_TEMPERATURE               0.090243
S_METALLICITY               0.092922
S_AGE                       0.110499
S_TIDAL_LOCK                0.136728
P_TEMP_EQUIL_MIN            0.140296
P_TEMP_EQUIL                0.141131
P_TPERI                     0.141285
P_TEMP_EQUIL_MAX            0.142674
P_HABZONE_CON               0.298875
P_ESI                       0.427036
P_HABZONE_OPT               0.491481
Name: P_HABITABLE, dtype: float64
```

Baseline Correlation Matrix



Top 25 Correlation Matrix



Label Encoding

```
objects = df.select_dtypes(include=['object'])
no_objects = df.select_dtypes(exclude=['object'])
no_objects.head()
```

```
objects.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4048 entries, 0 to 4047
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   P_NAME                4048 non-null   object
 1   P_UPDATED             4048 non-null   object
 2   P_DETECTION           4048 non-null   object
 3   S_NAME                4048 non-null   object
 4   S_ALT_NAMES           4048 non-null   object
 5   S_RA_T                4048 non-null   object
 6   S_DEC_T               4048 non-null   object
 7   S_CONSTELLATION       4048 non-null   object
 8   S_CONSTELLATION_ABR   4048 non-null   object
 9   S_CONSTELLATION_ENG   4048 non-null   object
```

```
df['P_HABITABLE'].corr(objects['S_CONSTELLATION'])
```

```
-0.009034930834177406
```

Feature Selection

Here we are checking the columns that contain objects to see if the data is worth encoding.

In [19]:
```
objects['P_DETECTION'].value_counts()
```

Out[19]:
```
Transit                          3114
Radial Velocity                   765
Microlensing                       78
Imaging                            47
Transit Timing Variations          18
Eclipse Timing Variations          11
Orbital Brightness Modulation       6
Pulsar Timing                       6
Pulsation Timing Variations         2
Astrometry                          1
Name: P_DETECTION, dtype: int64
```

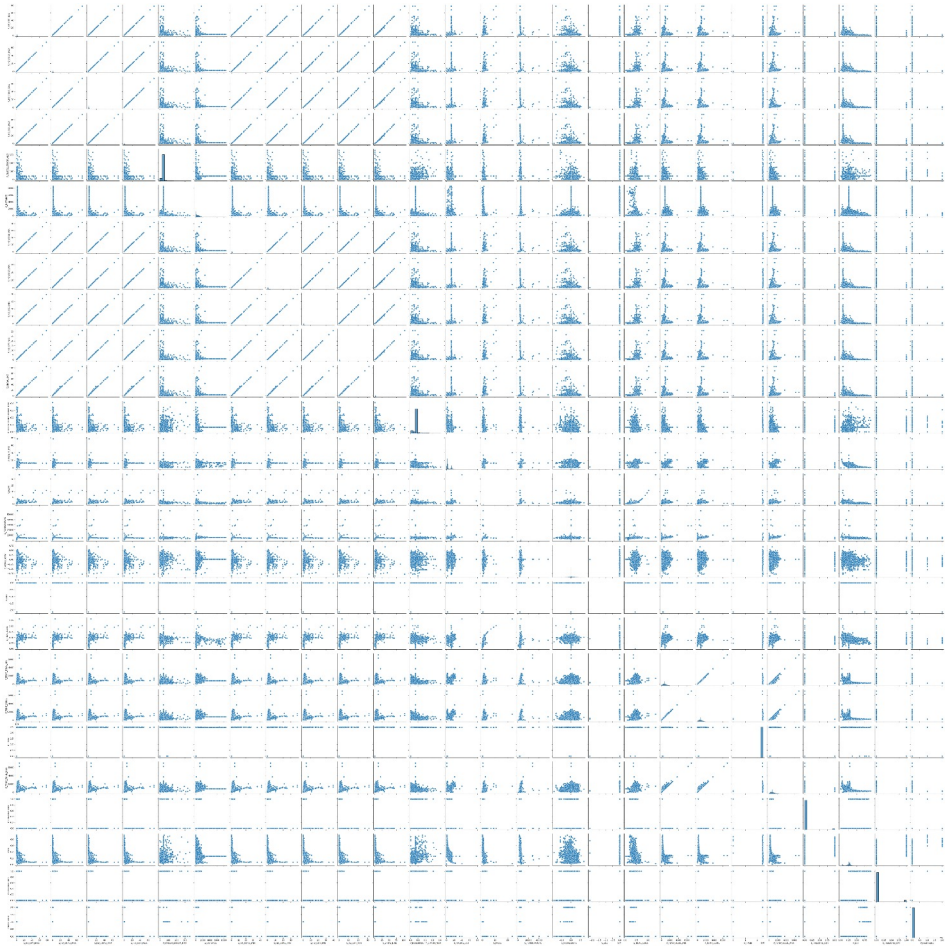
In [20]:
```
objects['S_RA_T'].value_counts()
```

Out[20]:
```
18 57 44.0416    8
23 06 29.2875    7
19 48 27.6332    6
19 44 27.0191    6
23 13 16.9820    6
                ..
18 56 23.4372    1
19 43 33.0061    1
19 18 59.3590    1
19 45 13.9115    1
12 20 43.0286    1
Name: S_RA_T, Length: 3010, dtype: int64
```

In [21]:
```
objects['S_DEC_T'].value_counts()
```

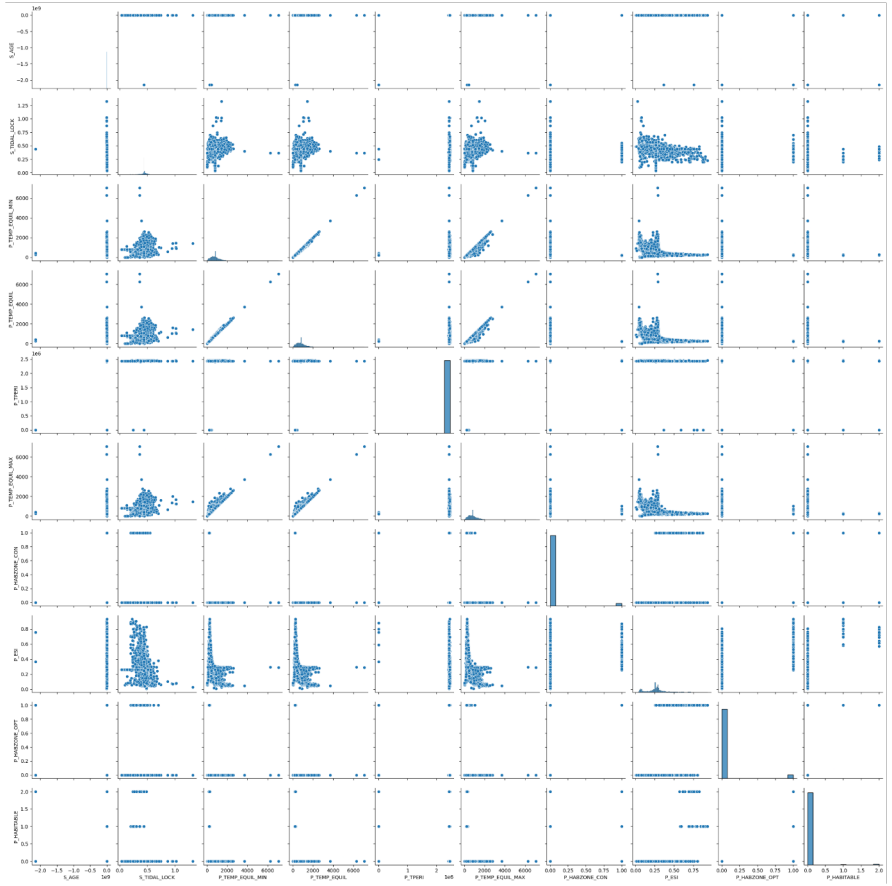
Out[21]:
```
+49 18 18.5796    8
-05 02 28.5864    7
+41 54 32.8968    6
+39 58 43.6044    6
+57 10 06.0744    6
                 ..
+47 48 38.2968    1
+43 09 21.8268    1
+39 31 24.4848    1
+43 49 16.8600    1
+17 47 34.3248    1
Name: S_DEC_T, Length: 3010, dtype: int64
```

```
# Removing data columns that are related to each other in the correlation matrix
skim_data = df_top25.copy()
skim_data = skim_data.drop(['S_HZ_OPT_MAX','S_HZ_CON1_MAX', 'S_HZ_CON0_MAX', 'S_HZ_CON_MAX',
                            'S_HZ_CON0_MIN', 'S_HZ_CON1_MIN', 'S_HZ_CON_MIN', 'S_HZ_OPT_MIN',
                            'P_TEMP_EQUIL_MIN', 'P_TEMP_EQUIL_MAX'], axis = 1)
```

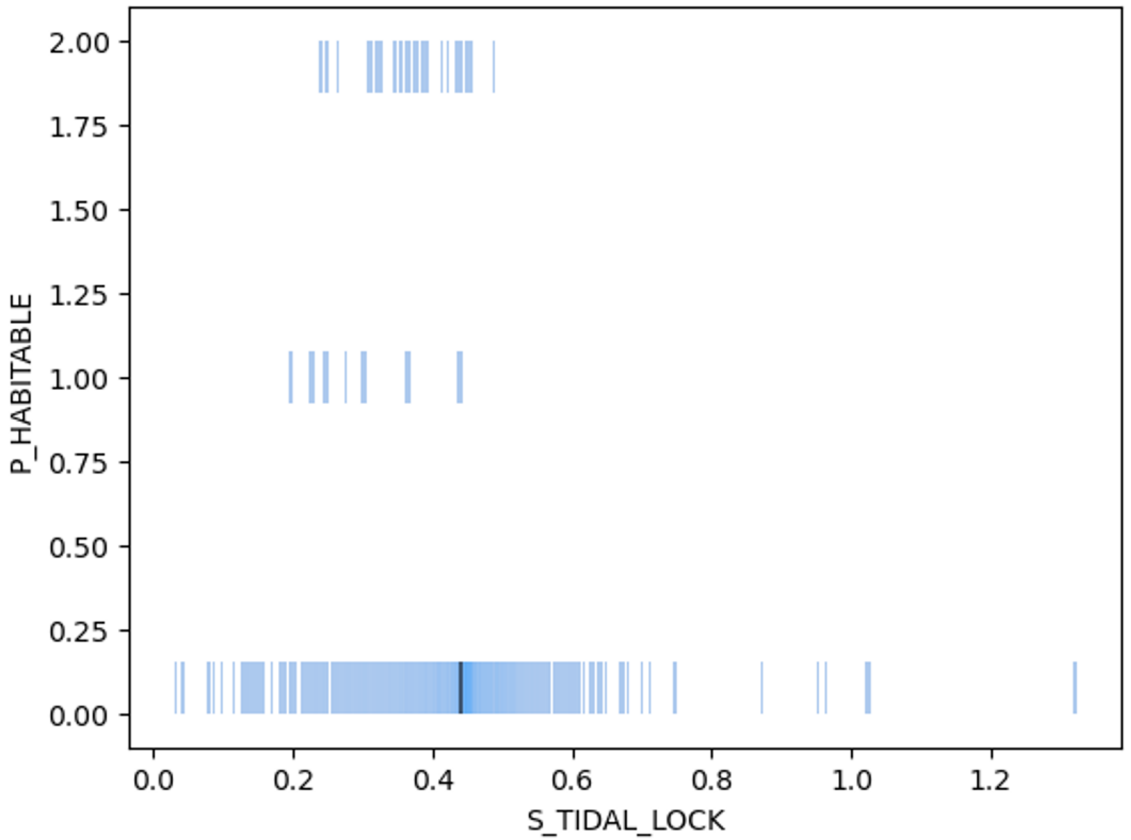


Pair-Plot

Pair-Plot Top 10



Histplot: Tidal Lock Habitability



Baseline Model

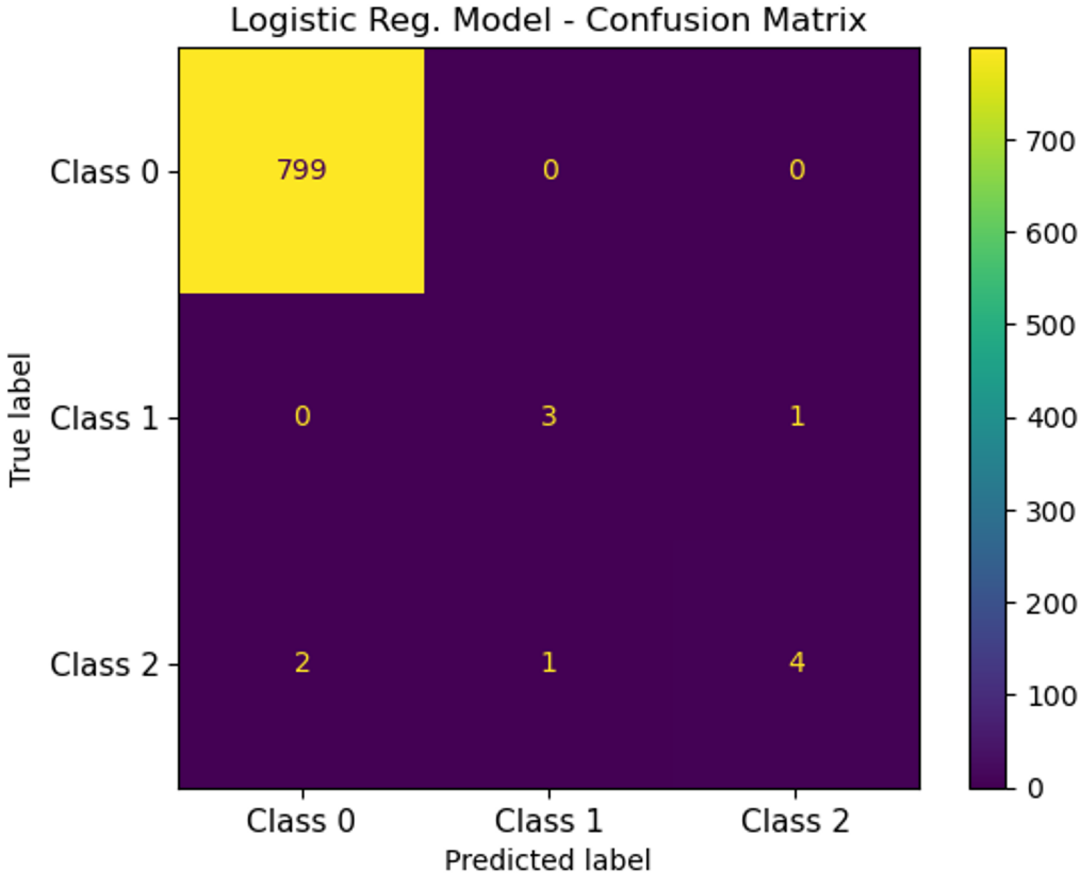
```
xtrain, xtest, ytrain, ytest =  train_test_split(x,y, test_size = 0.2, random_state=42, stratify=y)
```

```
from sklearn.linear_model import LogisticRegression

clf=LogisticRegression()
clf.fit(xtrain,ytrain)

#print score for test data
print(clf.score(xtest,ytest))
```

```
0.9950617283950617
```

```
from sklearn.metrics import ConfusionMatrixDisplay
cm = ConfusionMatrixDisplay.from_estimator(clf,xtest, ytest)
plt.title("Logistic Reg. Model - Confusion Matrix")
plt.xticks(range(3), ["Class 0","Class 1", "Class 2"], fontsize=11)
plt.yticks(range(3), ["Class 0","Class 1", "Class 2"], fontsize=11)
plt.show()
```

⇒



Logistic Regression

Logistic Regression on Top 25

Random Over Sampling (ROS)

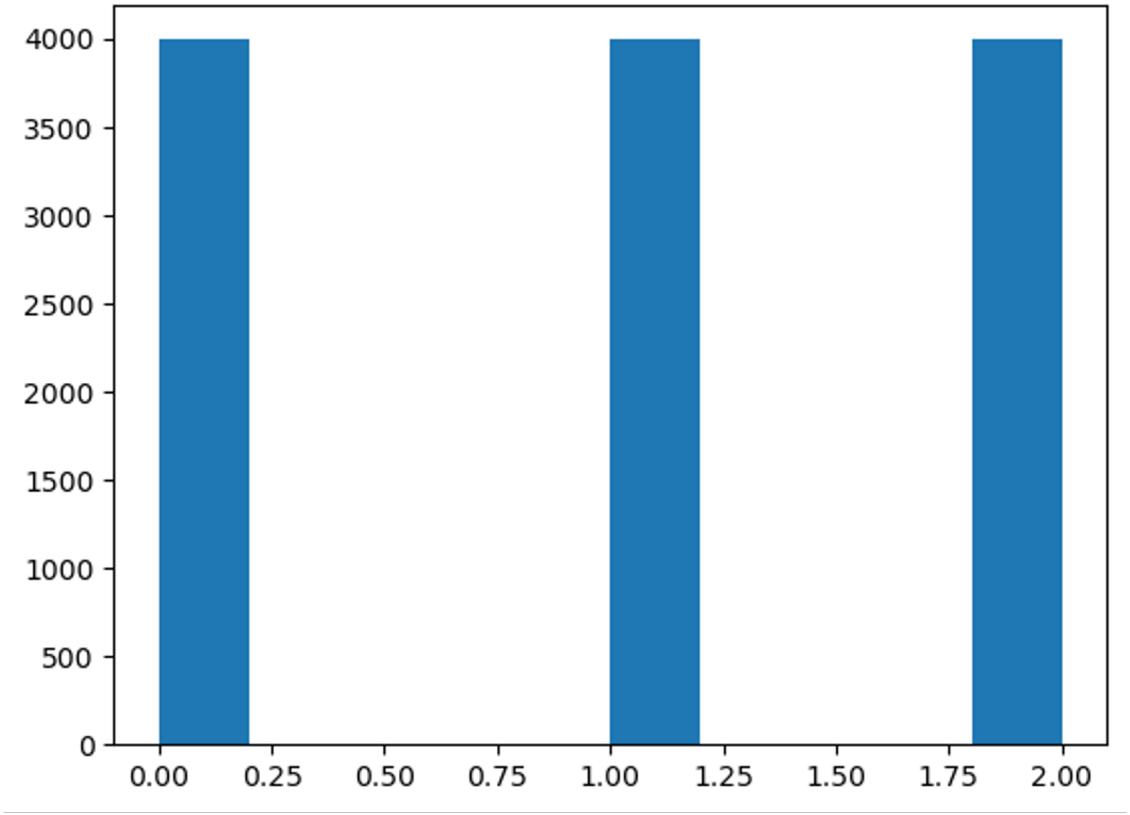
```
y2=skim_data['P_HABITABLE']
x2=skim_data.drop(['P_HABITABLE'],axis=1).copy()

# Balancing data values (Imbalanced data shown above)
ROS = RandomOverSampler(random_state=42,sampling_strategy='auto')
x_res, y_res = ROS.fit_resample(x2,y2)
# plt.hist(y_res)

print(y_res.value_counts())
```

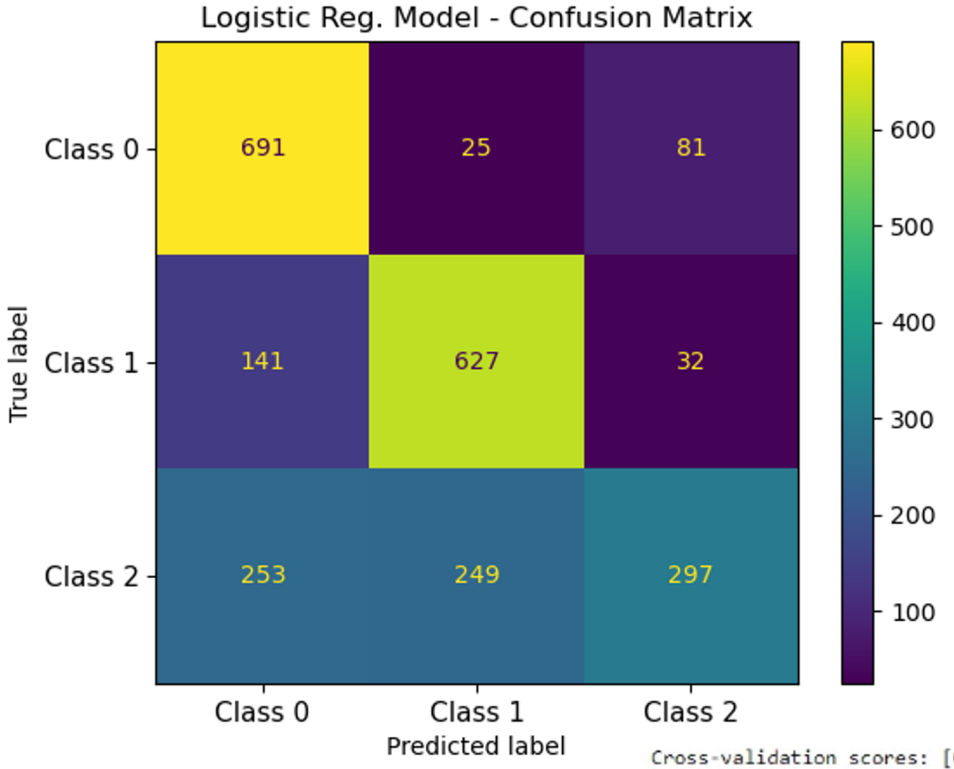
```
0    3993
2    3993
1    3993
Name: P_HABITABLE, dtype: int64
```

```
y22=skim_data['P_HABITABLE'].replace(2,1)

# Balancing data values (Imbalanced data shown above)
ROS = RandomOverSampler(random_state=42,sampling_strategy='auto')
x2_res, y2_res = ROS.fit_resample(x2,y22)
plt.hist(y_res)

print(y2_res.value_counts())
```

```
0    3993
1    3993
Name: P_HABITABLE, dtype: int64
```

Results of Random Oversampling

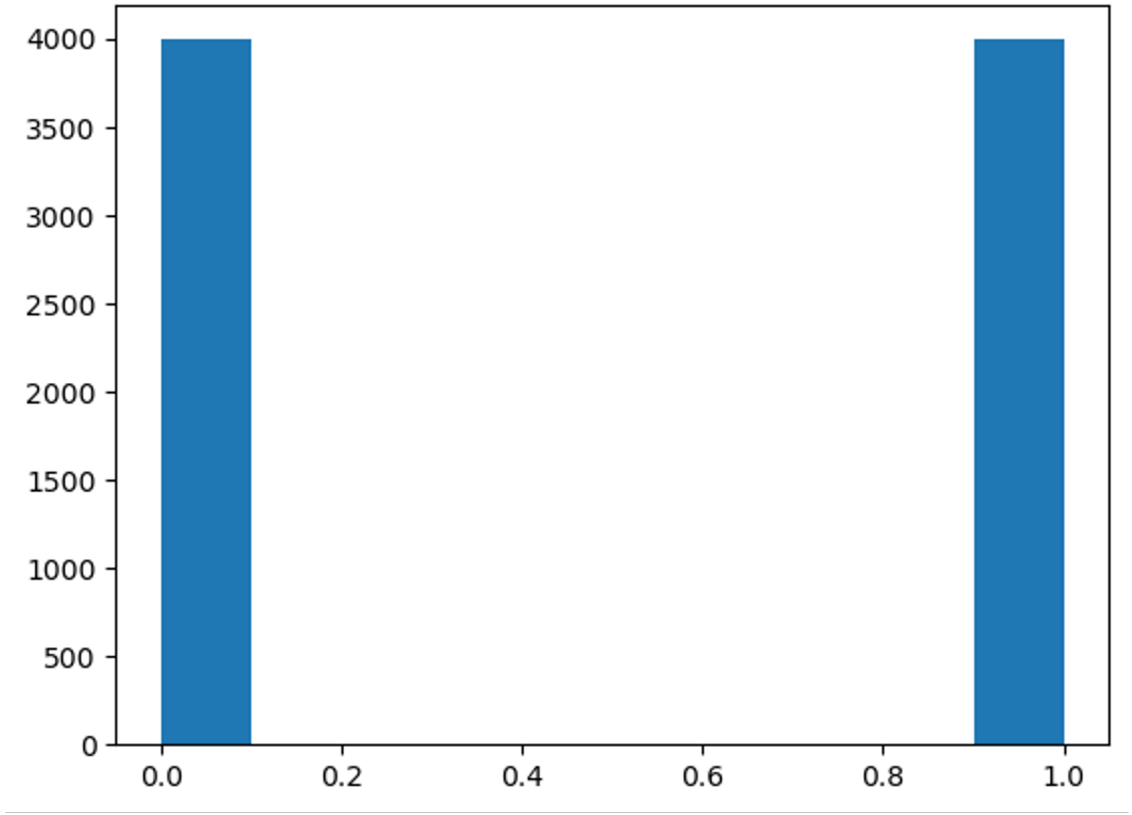




Logistic Regression

Logistic Regression after Random Oversampling

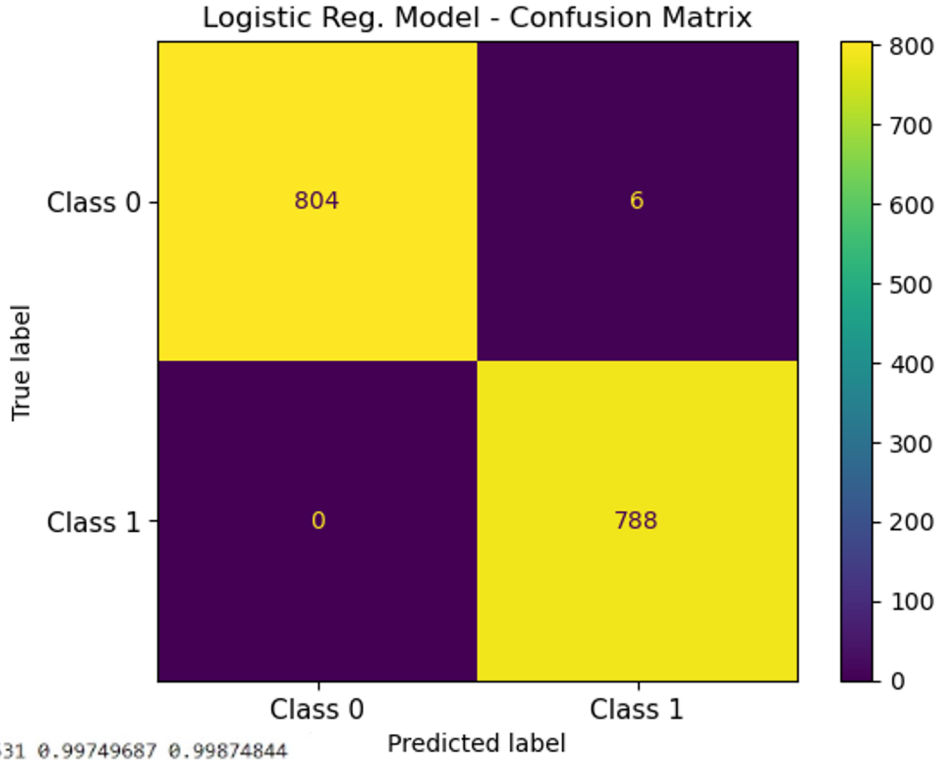
```
Cross-validation scores: [0.70617696 0.67195326 0.6836394  0.34557596 0.65859766 0.69949917
 0.34390651 0.34307179 0.75041736 0.67752715 0.34808013 0.67863105
 0.67278798 0.68280467 0.65776294 0.68948247 0.66944908 0.68196995
 0.57762938 0.67335004 0.65108514 0.67195326 0.34808013 0.67445743
 0.34390651 0.69449082 0.34474124 0.67863105 0.6803005  0.68421053]
Average cross-validation score: 0.60
```



Random Oversampling after combining 1 & 2

Logistic Regression

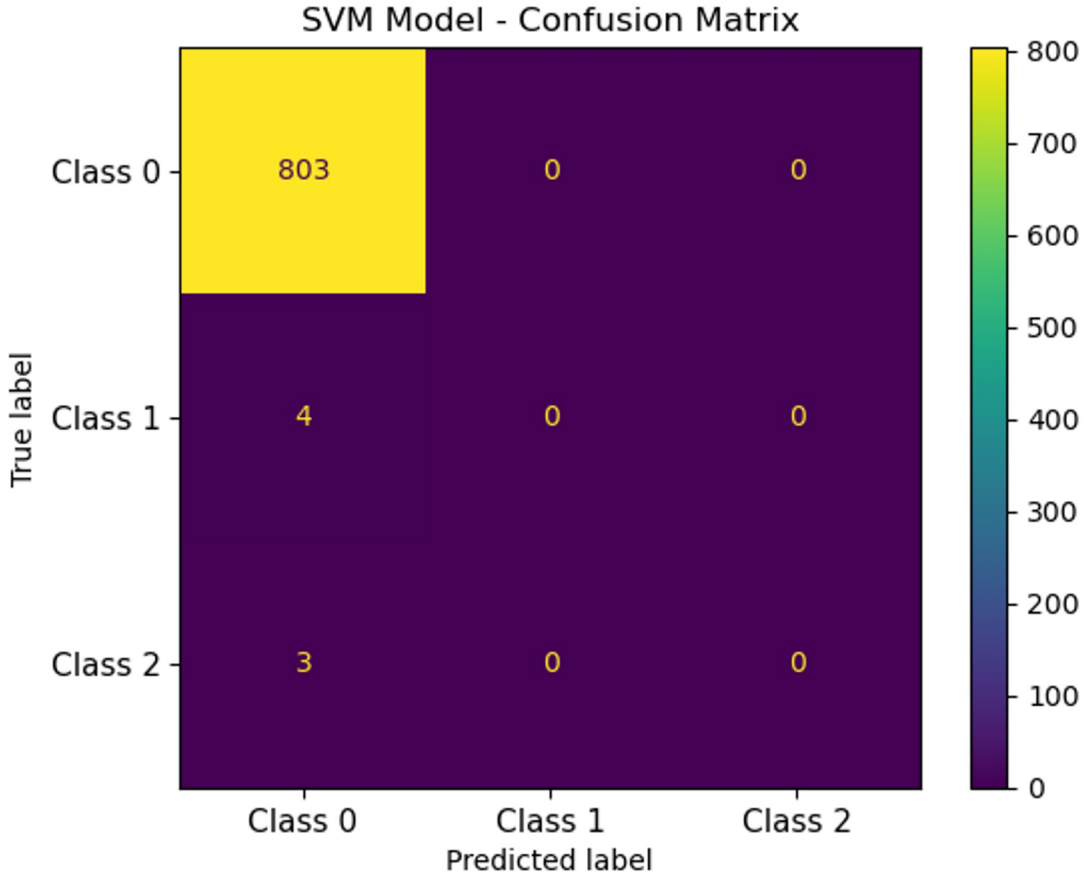
Logistic Regression after combining classifications 1 & 2

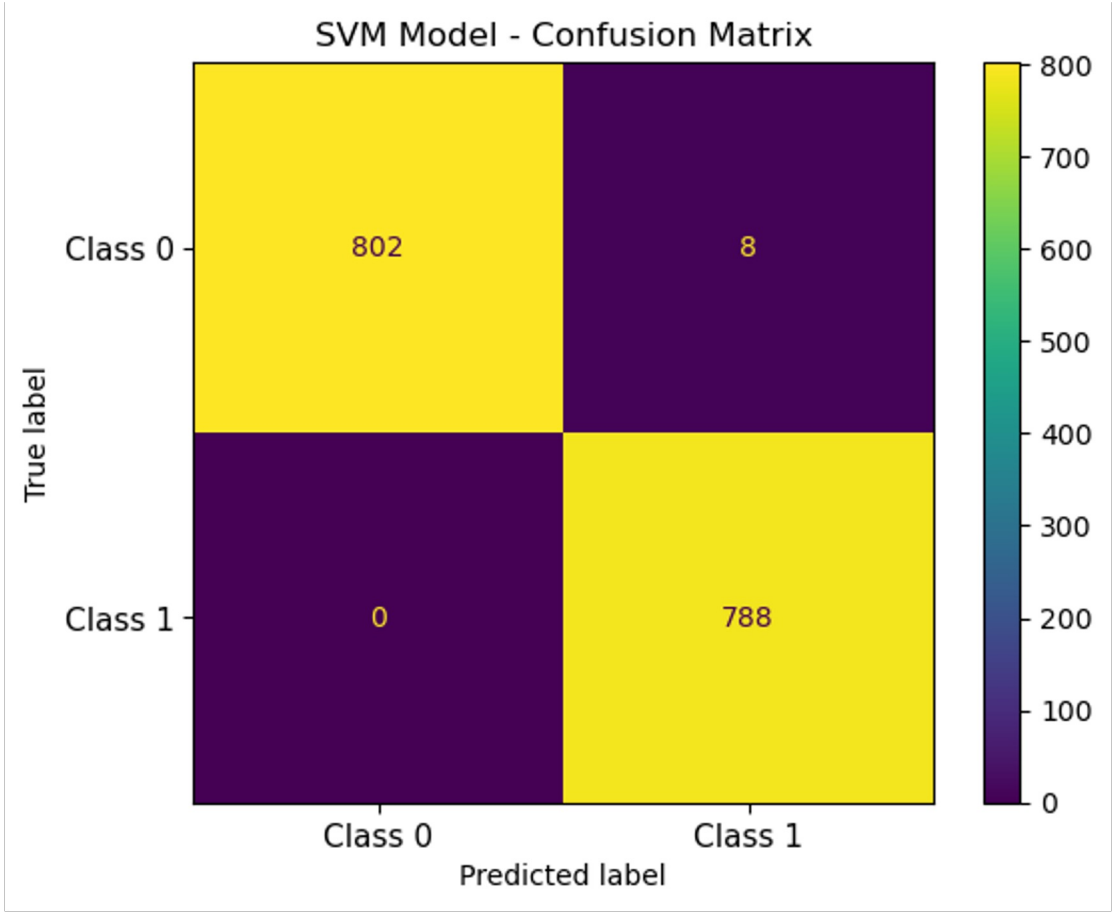



```
Cross-validation scores: [1.         1.         0.99874844 0.99624531 0.99749687 0.99874844
 0.99874687 0.99373434 0.99874687 0.9962406  1.         0.99749687
 1.         0.99749687 0.99374218 0.99874844 0.99874687 0.99749373
 0.99749373 0.99749373 0.99499374 1.         0.99874844 0.99874844
 0.99749687 0.99874844 0.99498747 0.99498747 0.99874687 1.        ]
Average cross-validation score: 0.9978
```

Support Vector Machine

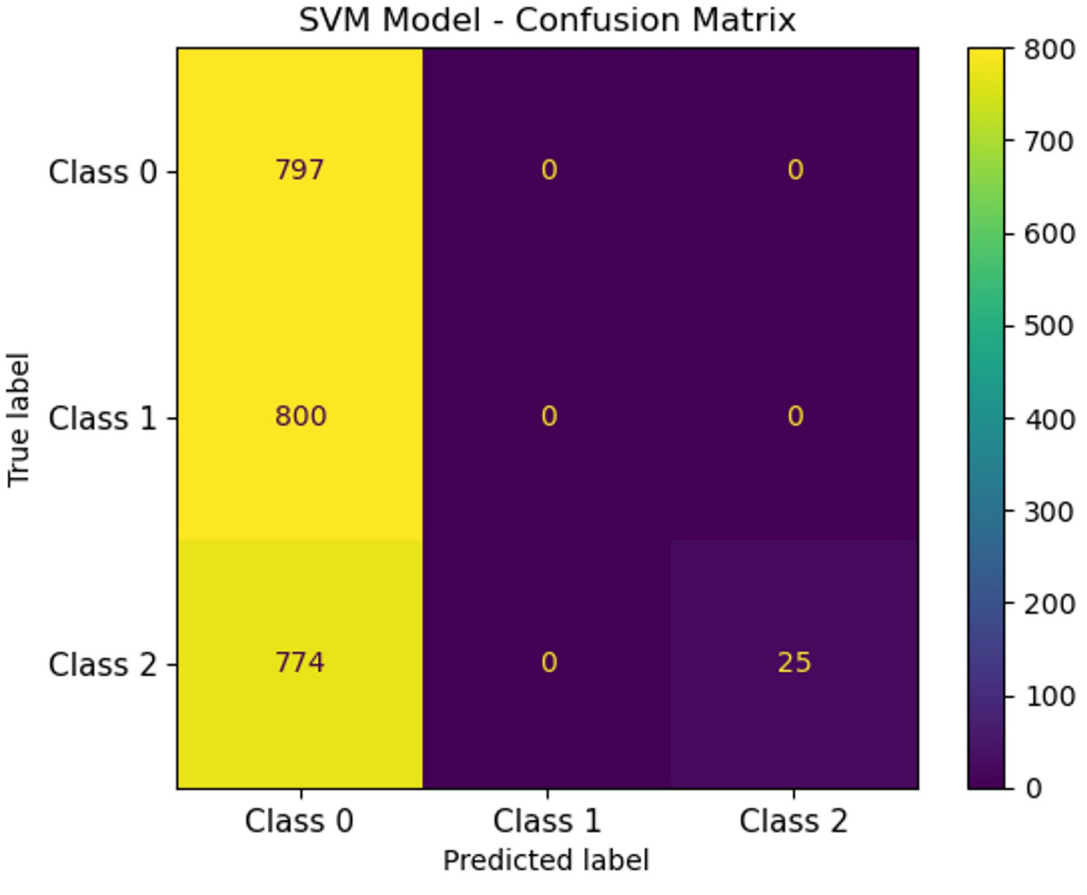
SVM on Top 25, before processing





Support Vector Machine

SVM after combining classifications 1 & 2



Support Vector Machine

SVM on Random Oversampled data

T-SNE (Imbalanced and Balanced)

```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 4048 samples in 0.001s...
[t-SNE] Computed neighbors for 4048 samples in 0.237s...
[t-SNE] Computed conditional probabilities for sample 1000 / 4048
[t-SNE] Computed conditional probabilities for sample 2000 / 4048
[t-SNE] Computed conditional probabilities for sample 3000 / 4048
[t-SNE] Computed conditional probabilities for sample 4000 / 4048
[t-SNE] Computed conditional probabilities for sample 4048 / 4048
[t-SNE] Mean sigma: 160.047582
[t-SNE] KL divergence after 250 iterations with early exaggeration: 64.023384
[t-SNE] KL divergence after 300 iterations: 1.347570
```

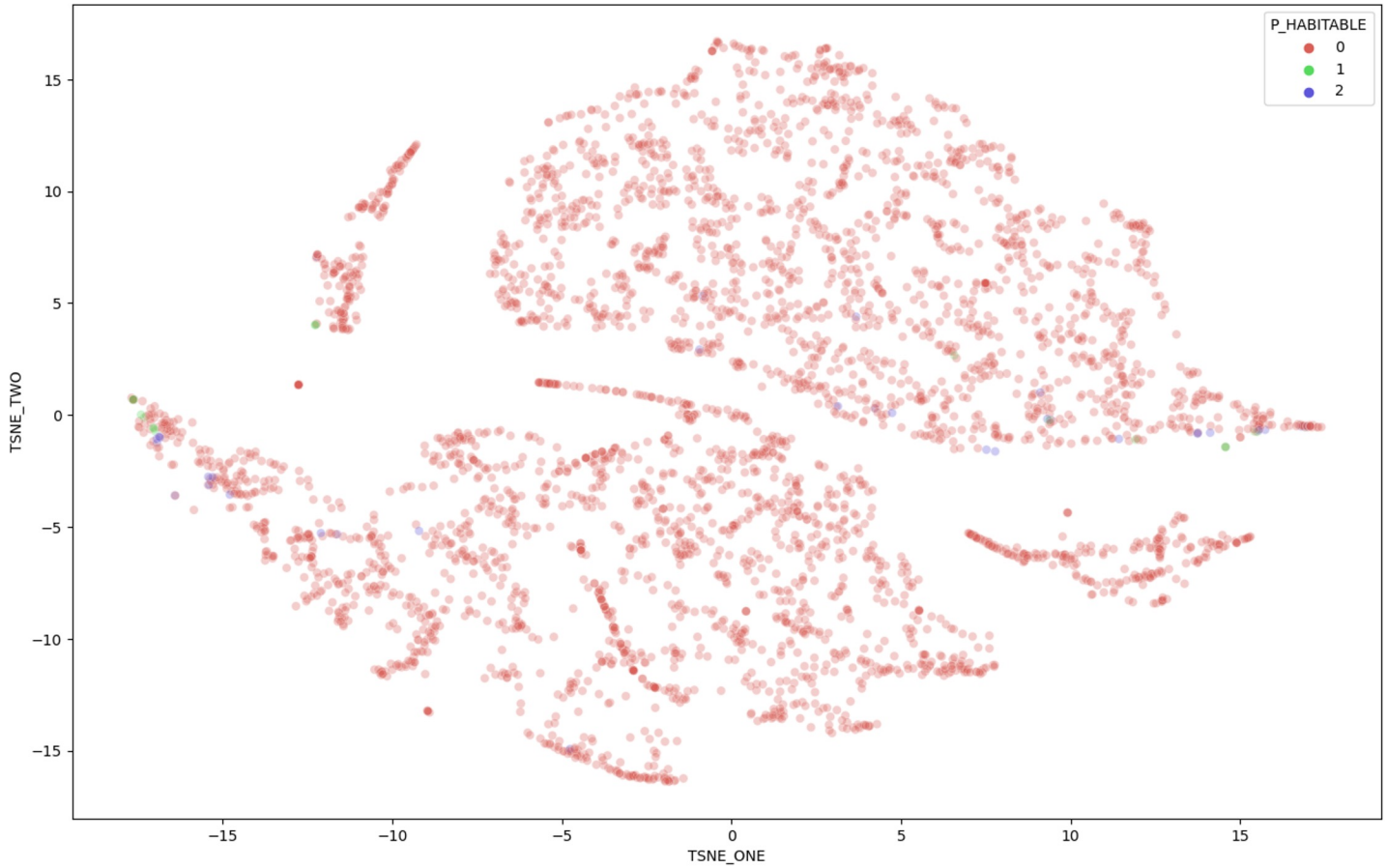
```
plt.figure(figsize=(16,10))
sns.scatterplot(
    x="TSNE_ONE", y="TSNE_TWO",
    hue="P_HABITABLE",
    palette=sns.color_palette("hls", 3),
    data=df_imbalanced,
    legend="full",
    alpha=0.3
)
```

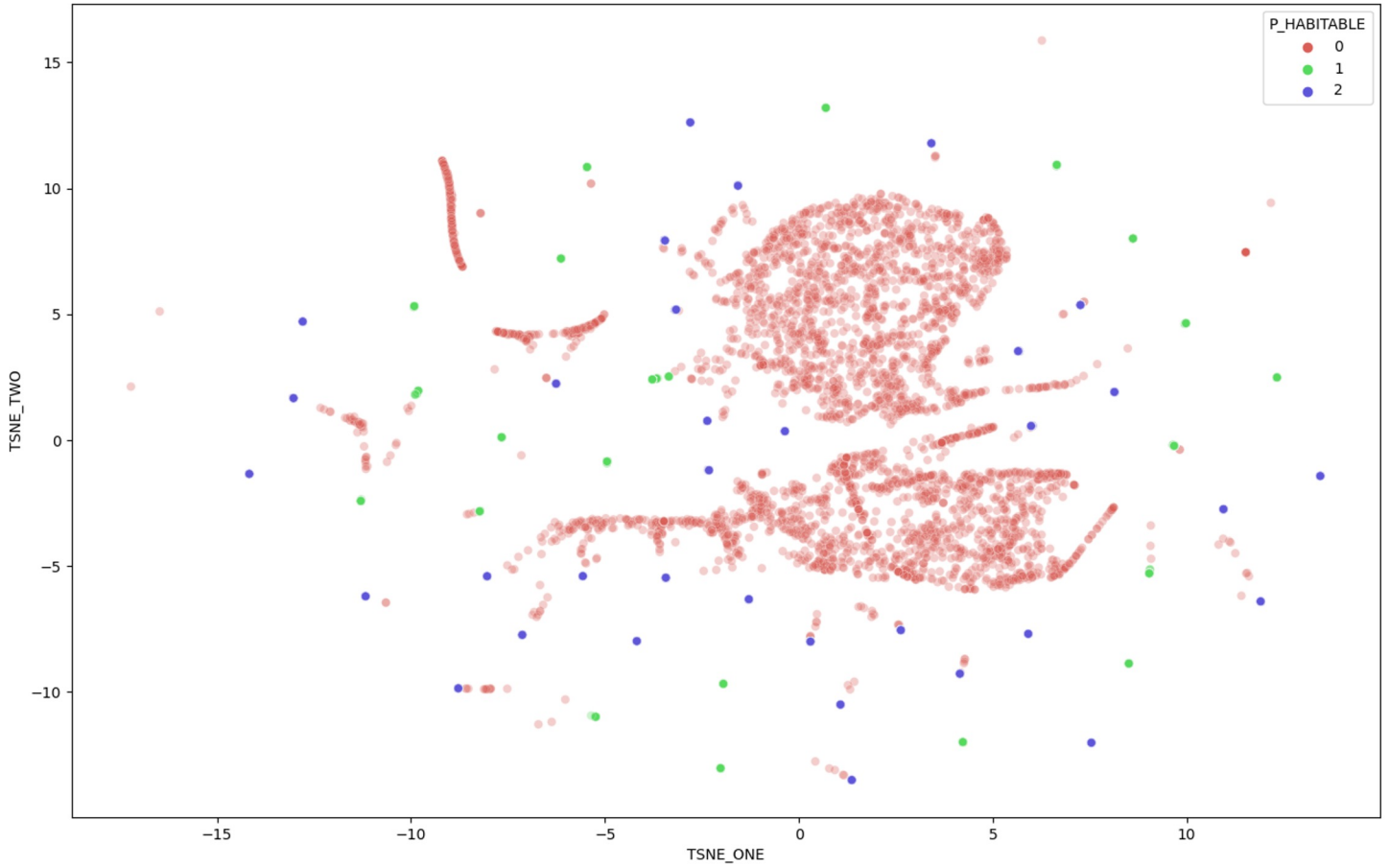
```
<AxesSubplot:xlabel='TSNE_ONE', ylabel='TSNE_TWO'>
```

```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 11979 samples in 0.030s...
[t-SNE] Computed neighbors for 11979 samples in 0.698s...
[t-SNE] Computed conditional probabilities for sample 1000 / 11979
[t-SNE] Computed conditional probabilities for sample 2000 / 11979
[t-SNE] Computed conditional probabilities for sample 3000 / 11979
[t-SNE] Computed conditional probabilities for sample 4000 / 11979
[t-SNE] Computed conditional probabilities for sample 5000 / 11979
[t-SNE] Computed conditional probabilities for sample 6000 / 11979
[t-SNE] Computed conditional probabilities for sample 7000 / 11979
[t-SNE] Computed conditional probabilities for sample 8000 / 11979
[t-SNE] Computed conditional probabilities for sample 9000 / 11979
[t-SNE] Computed conditional probabilities for sample 10000 / 11979
[t-SNE] Computed conditional probabilities for sample 11000 / 11979
[t-SNE] Computed conditional probabilities for sample 11979 / 11979
[t-SNE] Mean sigma: 0.000000
[t-SNE] KL divergence after 250 iterations with early exaggeration: 53.185787
[t-SNE] KL divergence after 300 iterations: 1.394255
```

```
plt.figure(figsize=(16,10))
sns.scatterplot(
    x="TSNE_ONE", y="TSNE_TWO",
    hue="P_HABITABLE",
    palette=sns.color_palette("hls", 3),
    data=df_balanced,
    legend="full",
    alpha=0.3
)
```

```
<AxesSubplot:xlabel='TSNE_ONE', ylabel='TSNE_TWO'>
```


P_HABITABLE
0
1
2
TSNE_TWO
TSNE_ONE
15
10
5
0
−5
−10
−15
−15
−10
−5
0
5
10
15


P_HABITABLE
0
1
2
TSNE_TWO
TSNE_ONE
15
10
5
0
−5
−10
−15
−10
−5
0
5
10

PCA Plots

```
pca = PCA(n_components=2)
pca_result = pca.fit_transform(skim_data)
df_imbalanced['pca-one'] = pca_result[:,0]
df_imbalanced['pca-two'] = pca_result[:,1]
```

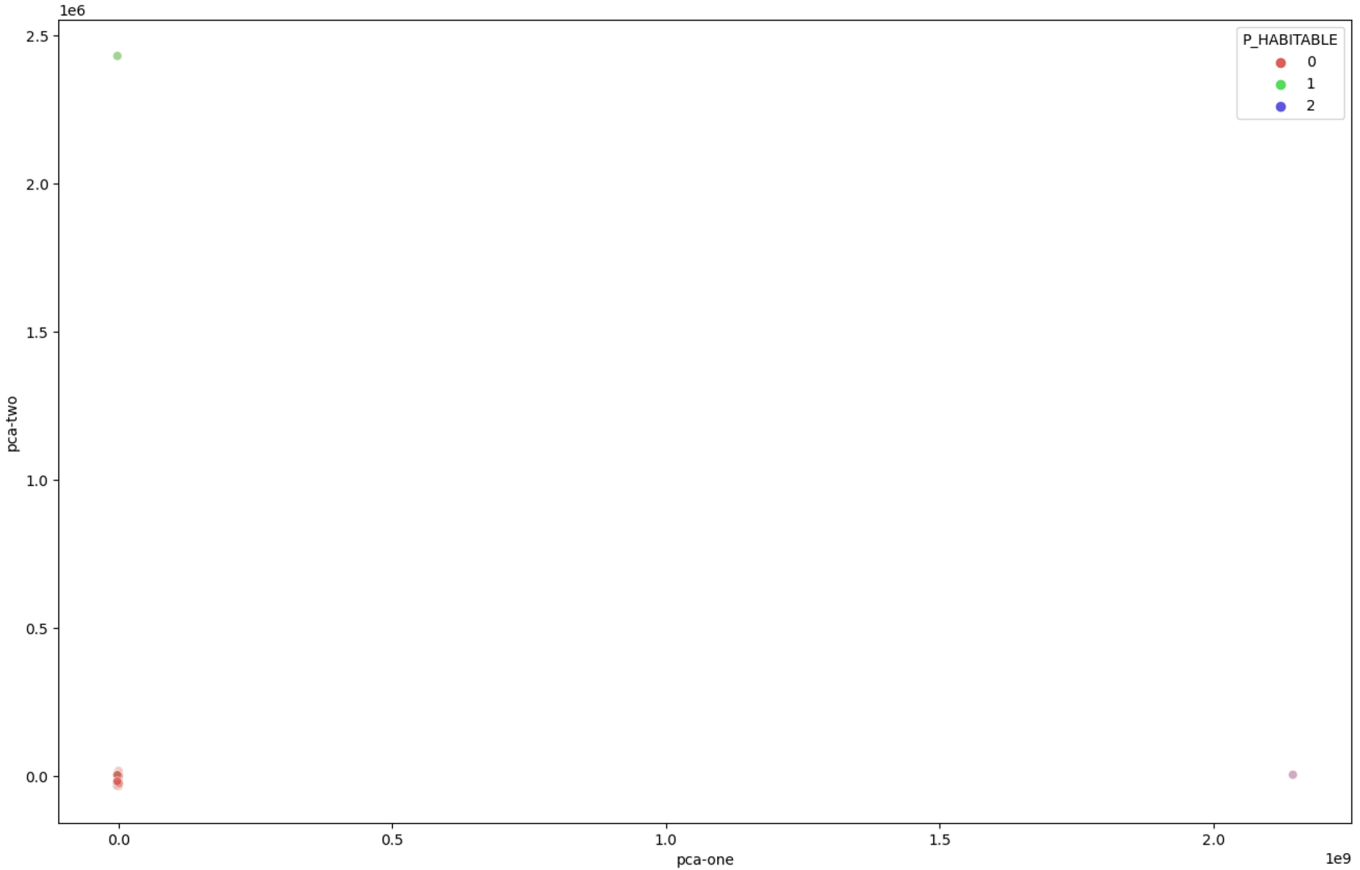
```
plt.figure(figsize=(16,10))
sns.scatterplot(
    x="pca-one", y="pca-two",
    hue="P_HABITABLE",
    palette=sns.color_palette("hls", 3),
    data=df_imbalanced,
    legend="full",
    alpha=0.3
)
```

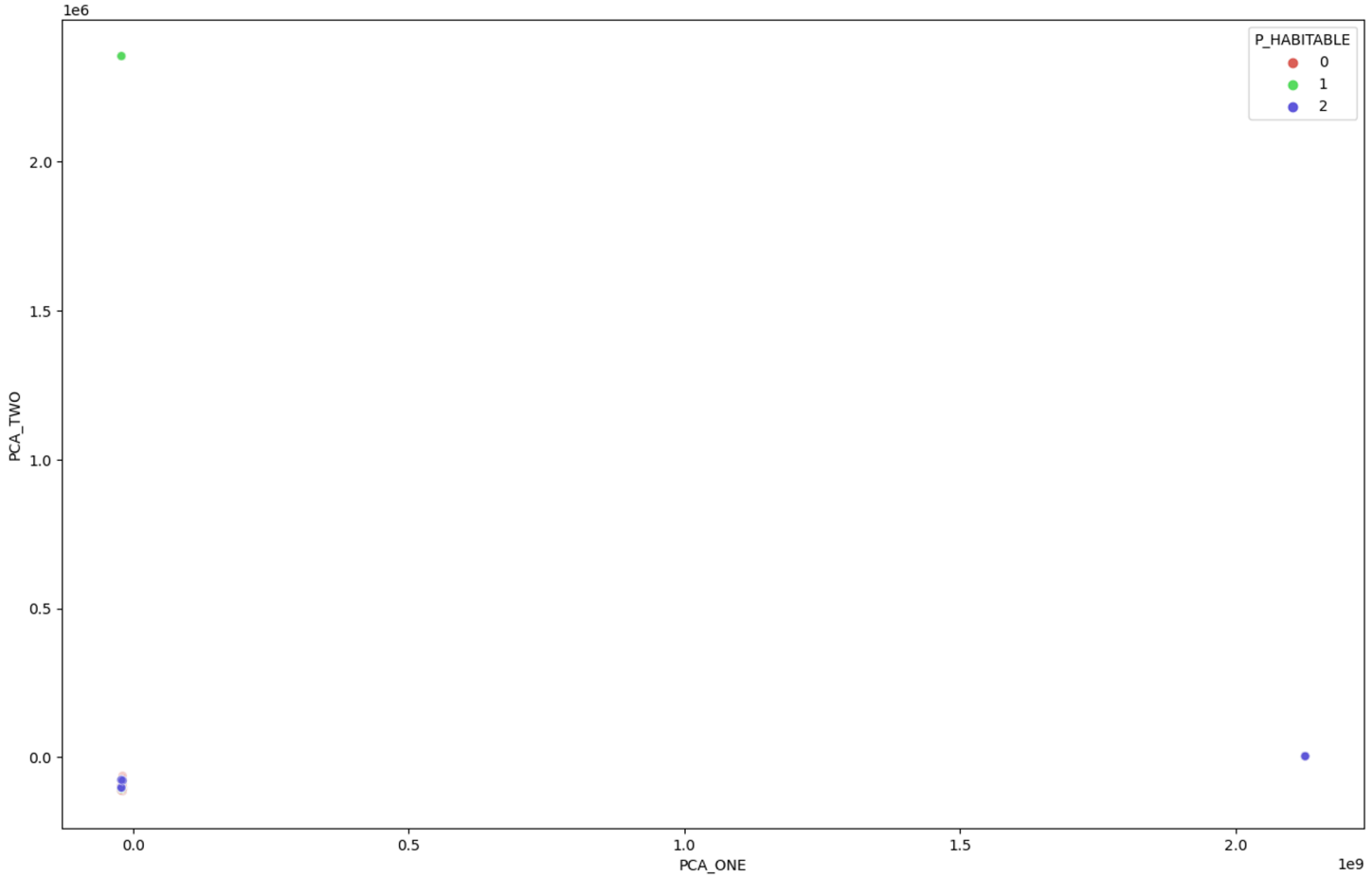
```
<AxesSubplot:xlabel='pca-one', ylabel='pca-two'>
```

```
pca = PCA(n_components=2)
pca_result_new = pca.fit_transform(X_res)
df_balanced=pd.DataFrame()

df_balanced['PCA_ONE']=pca_result_new[:,0]
df_balanced['PCA_TWO']=pca_result_new[:,1]
df_balanced['P_HABITABLE']=y_res
```

```
plt.figure(figsize=(16,10))
sns.scatterplot(
    x="PCA_ONE", y="PCA_TWO",
    hue="P_HABITABLE",
    palette=sns.color_palette("hls", 3),
    data=df_balanced,
    legend="full",
    alpha=0.3
)
```

```
<AxesSubplot:xlabel='PCA_ONE', ylabel='PCA_TWO'>
```


1e6
2.5
2.0
1.5
1.0
0.5
0.0
pca-two
P_HABITABLE
0
1
2
0.0
0.5
1.0
1.5
2.0
pca-one
1e9


1e6
P_HABITABLE
0
1
2
2.0
1.5
1.0
0.5
0.0
PCA_TWO
0.0
0.5
1.0
1.5
2.0
1e9
PCA_ONE