ara218 /
Ara218

Code Issues Pull requests Projects Wiki Security Insights S

main

Ara218 / HomeWork / HW_6.ipynb

ara218 homework 6 done History

1 contributor

1.74 MB

HW 6

This assignment covers all fundamental concepts required for completing a project

DO NOT ERASE MARKDOWN CELLS AND INSTRUCTIONS IN YOUR HW submission

- **Q** - QUESTION
- **A** - Where to input your answer

Instructions

Keep the following in mind for all notebooks you develop:

- Structure your notebook.
- Use headings with meaningful levels in Markdown cells, and explain the questions each piece of code is to answer or the reason it is there.
- Make sure your notebook can always be rerun from top to bottom.
- Please start working on this assignment as soon as possible. If you are a beginner in Python this might take a long time. One of the objectives of this assignment is to help you learn python and scikit-learn package.
- See README.md for homework submission instructions

Related Tutorials

Refreshers

- Intro to Machine Learning w scikit-learn
- A tutorial on statistical-learning for scientific data processing

Classification Approaches

- Logistic Regression with Sklearn
- KNN with sklearn
- Support Vector machine example
- SVC
- Bagging Classifier
- Gradient Boosting Classifier

Modeling

- Cross-validation
- Plot Confursion Matrix with Sklearn
- Confusion Matrix Display

Import all required library

In [4]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, MaxAbsScaler
import json
import lightgbm as lgbm
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import seaborn as sns
from imblearn.over_sampling import RandomOverSampler
from sklearn.ensemble import RandomForestClassifier
```

Data Processing

Q1 Get training data from the dataframe

1. Load `HW6_data.csv` from `data` folder into data frame
2. Print the head of the dataframe
3. Print the shape of the dataframe
4. Print the description of the dataframe
5. Check if the dataset has NULL values. (Show number of NULL values per column)
6. Assign `Cover_Type` values to Y
7. Assign rest of the column values to X

A1 Fill the cell blocks below, Create new cell as per your necessary

In [7]:

```
#You can create or remove cells as per your need
df = pd.read_csv('C:\\Users\\alsae\\Desktop\\fake\\2024Spring\\data\\HW6_data.c
print(df.head(5))
print(df.shape)
print(df.describe())
print(df.isnull().sum())
```

```
   Elevation  Aspect  Slope  Horizontal_Distance_To_Hydrology  \
0     3080.0     137   18.0                               166
1     2758.0      19    8.0                               551
2     2779.0      86    9.0                                43
3     2811.0     296    0.0                               287
4     2956.0     314   26.0                                71

   Vertical_Distance_To_Hydrology  Horizontal_Distance_To_Roadways  \
0                               1                             1009
1                              49                             1766
2                             -10                             3889
3                               4                              788
4                              22                             2910
```

```
4                              22                              2910

   Hillshade_9am  Hillshade_Noon  Hillshade_3pm  \
0          250.0             198            166
1          225.0             231            124
2          155.0             204            123
3          191.0             226            113
4          230.0             200             99

   Horizontal_Distance_To_Fire_Points  ...  Soil_Type32  Soil_Type33  \
0                              3635.0  ...            0            0
1                              1648.0  ...            0            0
2                               364.0  ...            0            0
3                               144.0  ...            0            0
4                               743.0  ...            0            0

   Soil_Type34  Soil_Type35  Soil_Type36  Soil_Type37  Soil_Type38  \
0            0          0.0            0            0            0
1            0          0.0            0            0            0
2            0          0.0            0            0            1
3            0          0.0            0            0            0
4            0          0.0            0            0            1

   Soil_Type39  Soil_Type40  Cover_Type
0            0          0.0           1
1            0          0.0           2
2            0          0.0           2
3            0          0.0           2
4            0          NaN           2

[5 rows x 55 columns]
(80000, 55)
          Elevation        Aspect         Slope  \
count  79433.000000  80000.000000  79346.000000
mean    2981.436531    151.634175     15.092494
std      287.979705    109.945631      8.528153
min     1813.000000    -29.000000     -3.000000
25%     2762.000000     60.000000      9.000000
50%     2967.000000    122.000000     14.000000
75%     3217.000000    246.000000     20.000000
max     4271.000000    400.000000     61.000000

       Horizontal_Distance_To_Hydrology  Vertical_Distance_To_Hydrology  \
count                      80000.000000                    80000.000000
mean                         271.564212                       51.510737
std                          227.532197                       68.091489
min                          -43.000000                     -276.000000
25%                          111.000000                        4.000000
50%                          212.000000                       31.000000
75%                          361.000000                       78.000000
max                         1544.000000                      562.000000

       Horizontal_Distance_To_Roadways  Hillshade_9am  Hillshade_Noon  \
count                     80000.000000   79200.000000    80000.000000
mean                       1770.080712     211.786818      221.069125
std                        1318.661060      30.822278       22.191030
min                        -238.000000      10.000000       69.000000
25%                         821.000000     198.000000      210.000000
50%                        1440.000000     218.000000      224.000000
75%                        2366.000000     233.000000      237.000000
max                        7604.000000     293.000000      264.000000
```

```
       Hillshade_3pm  Horizontal_Distance_To_Fire_Points  ...   Soil_Type32  \
count   80000.000000                        78870.000000  ...  80000.000000
mean      140.711750                         1578.058615  ...      0.038150
std        43.859689                         1125.780446  ...      0.191559
min       -48.000000                         -218.000000  ...      0.000000
25%       115.000000                          782.000000  ...      0.000000
50%       142.000000                         1362.000000  ...      0.000000
75%       169.000000                         2082.000000  ...      0.000000
max       268.000000                         8011.000000  ...      1.000000

        Soil_Type33   Soil_Type34   Soil_Type35   Soil_Type36   Soil_Type37  \
count  80000.000000  80000.000000  79720.000000  80000.000000  80000.000000
mean       0.037687      0.011838      0.015429      0.010812      0.012538
std        0.190441      0.108155      0.123252      0.103420      0.111268
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.000000      0.000000      0.000000
50%        0.000000      0.000000      0.000000      0.000000      0.000000
75%        0.000000      0.000000      0.000000      0.000000      0.000000
max        1.000000      1.000000      1.000000      1.000000      1.000000

        Soil_Type38   Soil_Type39   Soil_Type40    Cover_Type
count  80000.000000  80000.000000  75000.000000  80000.000000
mean       0.040325      0.039163      0.030707      1.770725
std        0.196722      0.193983      0.172523      0.892577
min        0.000000      0.000000      0.000000      1.000000
25%        0.000000      0.000000      0.000000      1.000000
50%        0.000000      0.000000      0.000000      2.000000
75%        0.000000      0.000000      0.000000      2.000000
max        1.000000      1.000000      1.000000      7.000000

[8 rows x 55 columns]
Elevation                              567
Aspect                                   0
Slope                                  654
Horizontal_Distance_To_Hydrology         0
Vertical_Distance_To_Hydrology           0
Horizontal_Distance_To_Roadways          0
Hillshade_9am                          800
Hillshade_Noon                           0
Hillshade_3pm                            0
Horizontal_Distance_To_Fire_Points    1130
Wilderness_Area1                         0
Wilderness_Area2                         0
Wilderness_Area3                         0
Wilderness_Area4                         0
Soil_Type1                               0
Soil_Type2                               0
Soil_Type3                               0
Soil_Type4                               0
Soil_Type5                               0
Soil_Type6                               0
Soil_Type7                               0
Soil_Type8                               0
Soil_Type9                               0
Soil_Type10                              0
Soil_Type11                              0
Soil_Type12                              0
Soil_Type13                              0
Soil_Type14                              0
```

```
Soil_Type15                           0
Soil_Type16                           0
Soil_Type17                           0
Soil_Type18                           0
Soil_Type19                           0
Soil_Type20                           0
Soil_Type21                           0
Soil_Type22                           0
Soil_Type23                           0
Soil_Type24                           0
Soil_Type25                           0
Soil_Type26                           0
Soil_Type27                           0
Soil_Type28                           0
Soil_Type29                           0
Soil_Type30                           0
Soil_Type31                           0
Soil_Type32                           0
Soil_Type33                           0
Soil_Type34                           0
Soil_Type35                         280
Soil_Type36                           0
Soil_Type37                           0
Soil_Type38                           0
Soil_Type39                           0
Soil_Type40                        5000
Cover_Type                            0
dtype: int64
```

In [8]:

```
X=df.drop(columns=['Cover_Type'])
Y=df['Cover_Type']
```

Q2: Open-Ended Questions: Observe the range of all feature values and statistical information from the dataframe description above.

1. If the dataset has NULL values, Give proper justification about the methods you will use to replace NULL values for specific columns.
2. Do you think in our dataset normalization is required? -- Give proper justification based on your opinion.
3. What type of normalization/Scaling technique you whould recommend for our dataset?

A2

```
Answer 1:

Answer 2:

Answer 3:
```

Q3:

1. Replace the null values with the best possible methods from your above observation
2. Use the above mentioned normalization technique on our HW_6 dataset.

3. Transform the X dataframe using choosen normalization technique.

`Note:` Make sure the scaled X has all column name same as `X dataframe`

A3 Fill the cell blocks below, Create new cell as per your necessary

In [9]:
```
for col in X.columns:
    if X[col].dtype == 'object':
        X[col] = X[col].fillna(X[col].mode()[0])
    else:
        X[col] = X[col].fillna(X[col].mean())
```

In [10]:
```
from sklearn.preprocessing import MinMaxScaler

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the data
Scaled_X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
```

In [11]:
```
Scaled_X
```

Out[11]:

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distanc
0	0.515460	0.386946	0.328125	0.131695	
1	0.384459	0.111888	0.171875	0.374291	
2	0.393002	0.268065	0.187500	0.054190	
3	0.406021	0.757576	0.046875	0.207940	
4	0.465012	0.799534	0.453125	0.071834	
...	...	...	...	...	
79995	0.288853	0.188811	0.171875	0.080655	
79996	0.370627	0.100233	0.218750	0.261500	
79997	0.557771	0.552448	0.109375	0.335224	
79998	0.635883	0.081585	0.359375	0.299937	
79999	0.567535	0.606061	0.234375	0.101449	

80000 rows × 54 columns

Q4:

1. Check again and show if there is any null values left in our `Scaled_X`.
2. Print all unique values/ different class id from the `Y data`.

A4 Fill the cell blocks below, Create new cell as per your necessary

In [12]:

```
print("NULL values in Scaled_X:")
print(Scaled_X.isnull().sum())
```

```
NULL values in Scaled_X:
Elevation                             0
Aspect                                0
Slope                                 0
Horizontal_Distance_To_Hydrology      0
Vertical_Distance_To_Hydrology        0
Horizontal_Distance_To_Roadways       0
Hillshade_9am                         0
Hillshade_Noon                        0
Hillshade_3pm                         0
Horizontal_Distance_To_Fire_Points    0
Wilderness_Area1                      0
Wilderness_Area2                      0
Wilderness_Area3                      0
Wilderness_Area4                      0
Soil_Type1                            0
Soil_Type2                            0
Soil_Type3                            0
Soil_Type4                            0
Soil_Type5                            0
Soil_Type6                            0
Soil_Type7                            0
Soil_Type8                            0
Soil_Type9                            0
Soil_Type10                           0
Soil_Type11                           0
Soil_Type12                           0
Soil_Type13                           0
Soil_Type14                           0
Soil_Type15                           0
Soil_Type16                           0
Soil_Type17                           0
Soil_Type18                           0
Soil_Type19                           0
Soil_Type20                           0
Soil_Type21                           0
Soil_Type22                           0
Soil_Type23                           0
Soil_Type24                           0
Soil_Type25                           0
Soil_Type26                           0
Soil_Type27                           0
Soil_Type28                           0
Soil_Type29                           0
Soil_Type30                           0
Soil_Type31                           0
Soil_Type32                           0
Soil_Type33                           0
Soil_Type34                           0
Soil_Type35                           0
Soil_Type36                           0
Soil_Type37                           0
Soil_Type38                           0
```

```
Soil_Type38                          0
Soil_Type39                          0
Soil_Type40                          0
dtype: int64
```

In [13]:
```
# Print unique values from Y
print("\nUnique values in Y:")
print(Y.unique())
```

```
Unique values in Y:
[1 2 3 7 6 4]
```

Part 1: Use a subset of whole data(N=20000) for Data Visualization

Data Subset Creation

1. First we are Selecting `N=20000` random rows from our original dataset which is `df` and create a new subset of data.

2. Using the below **rndperm** and selecting first N index from the `Scaled_X` and `Y`

In [16]:
```
np.random.seed(42)
rndperm = np.random.permutation(df.shape[0])
N = 20000
data_subset_x = Scaled_X.loc[rndperm[:N],:].copy()
data_subset_y = Y.loc[rndperm[:N]].copy()
```

Q5:

1. Use PCA and reduce the dimension of the **data_subset_x** into `3` .
2. Store the PCA reuslt into `pca_result` variable
3. Add the resutls from the PCA into the **data_subset_x** as new columns. (Choose any meaningful names for the columns)

A5 Fill the below cells. Use extra cells as per your necessary

In [17]:
```
#You can create or remove cells as per your need
from sklearn.decomposition import PCA

pca = PCA(n_components=3)
pca_result = pca.fit_transform(data_subset_x)
```

In [18]:
```
data_subset_x['PCA_Component_1'] = pca_result[:, 0]
data_subset_x['PCA_Component_2'] = pca_result[:, 1]
data_subset_x['PCA_Component_3'] = pca_result[:, 2]
```

Q6:

1. Use TSNE and reduce the dimension of the **data_subset_x** into `2` .

2. Store the TSNE reuslt into `tsne_results` variable
3. Add the resutls from the T-SNE into the **data_subset_x** as new columns. (Choose any meaningful names for the columns)

`Note:`

1. You can use `from sklearn.manifold import TSNE` for TSNE initialization.
2. Give value of n_components as per the question.
3. Also use other parameters while TSNE initialization as, `verbose=1, perplexity=40, n_iter=300`

A6 Fill the below cells. Use extra cells as per your necessary

In [19]:
```
#You can create or remove cells as per your need
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300)
tsne_results = tsne.fit_transform(data_subset_x)
```

```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 20000 samples in 0.006s...
[t-SNE] Computed neighbors for 20000 samples in 1.386s...
[t-SNE] Computed conditional probabilities for sample 1000 / 20000
[t-SNE] Computed conditional probabilities for sample 2000 / 20000
[t-SNE] Computed conditional probabilities for sample 3000 / 20000
[t-SNE] Computed conditional probabilities for sample 4000 / 20000
[t-SNE] Computed conditional probabilities for sample 5000 / 20000
[t-SNE] Computed conditional probabilities for sample 6000 / 20000
[t-SNE] Computed conditional probabilities for sample 7000 / 20000
[t-SNE] Computed conditional probabilities for sample 8000 / 20000
[t-SNE] Computed conditional probabilities for sample 9000 / 20000
[t-SNE] Computed conditional probabilities for sample 10000 / 20000
[t-SNE] Computed conditional probabilities for sample 11000 / 20000
[t-SNE] Computed conditional probabilities for sample 12000 / 20000
[t-SNE] Computed conditional probabilities for sample 13000 / 20000
[t-SNE] Computed conditional probabilities for sample 14000 / 20000
[t-SNE] Computed conditional probabilities for sample 15000 / 20000
[t-SNE] Computed conditional probabilities for sample 16000 / 20000
[t-SNE] Computed conditional probabilities for sample 17000 / 20000
[t-SNE] Computed conditional probabilities for sample 18000 / 20000
[t-SNE] Computed conditional probabilities for sample 19000 / 20000
[t-SNE] Computed conditional probabilities for sample 20000 / 20000
[t-SNE] Mean sigma: 0.184852
[t-SNE] KL divergence after 250 iterations with early exaggeration: 80.914803
[t-SNE] KL divergence after 300 iterations: 2.991430
```

In [20]:
```
data_subset_x['tSNE_Component_1'] = tsne_results[:, 0]
data_subset_x['tSNE_Component_2'] = tsne_results[:, 1]
```

Q7:

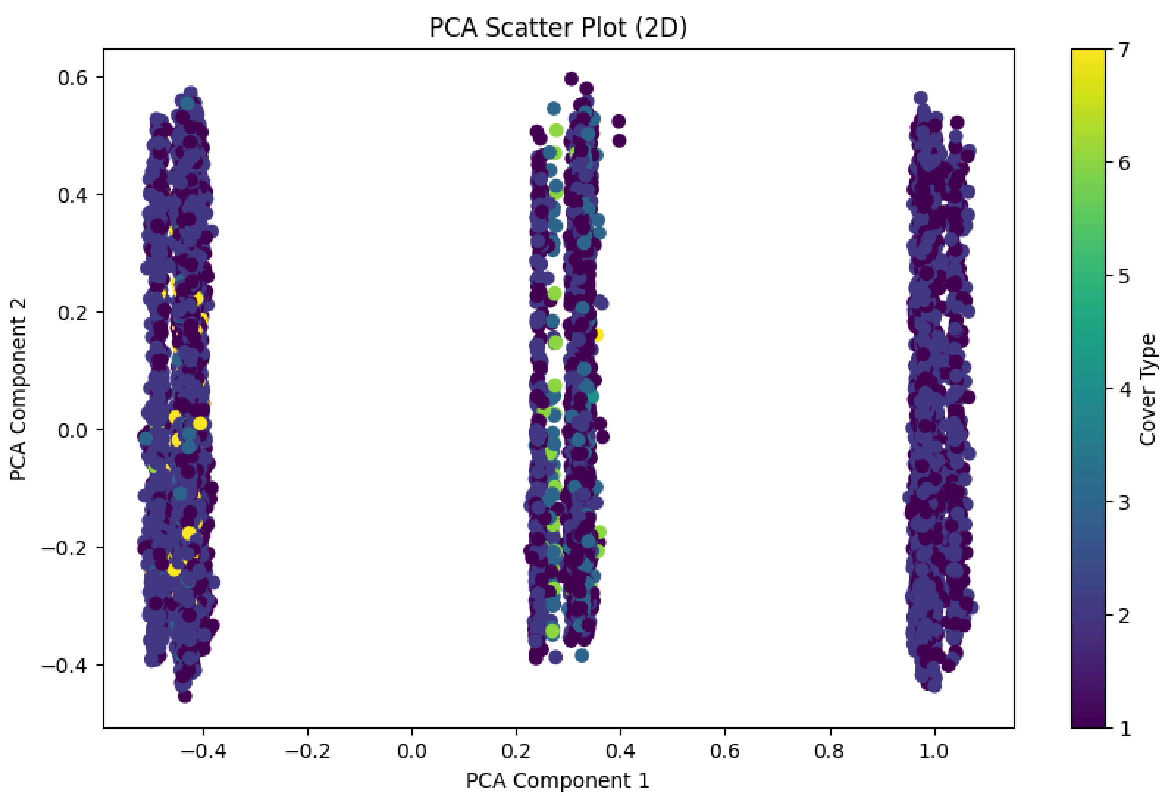
1. Create a new dataframe with name `df_plot`
2. This dataframe will merge everything from **data_subset_x** and **data_subset_y**
3. We need to give a name for the `data_subset_y` column. Use `Cover_Type` as the

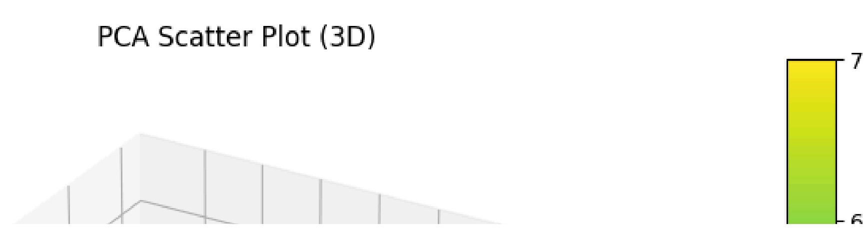
name of the column

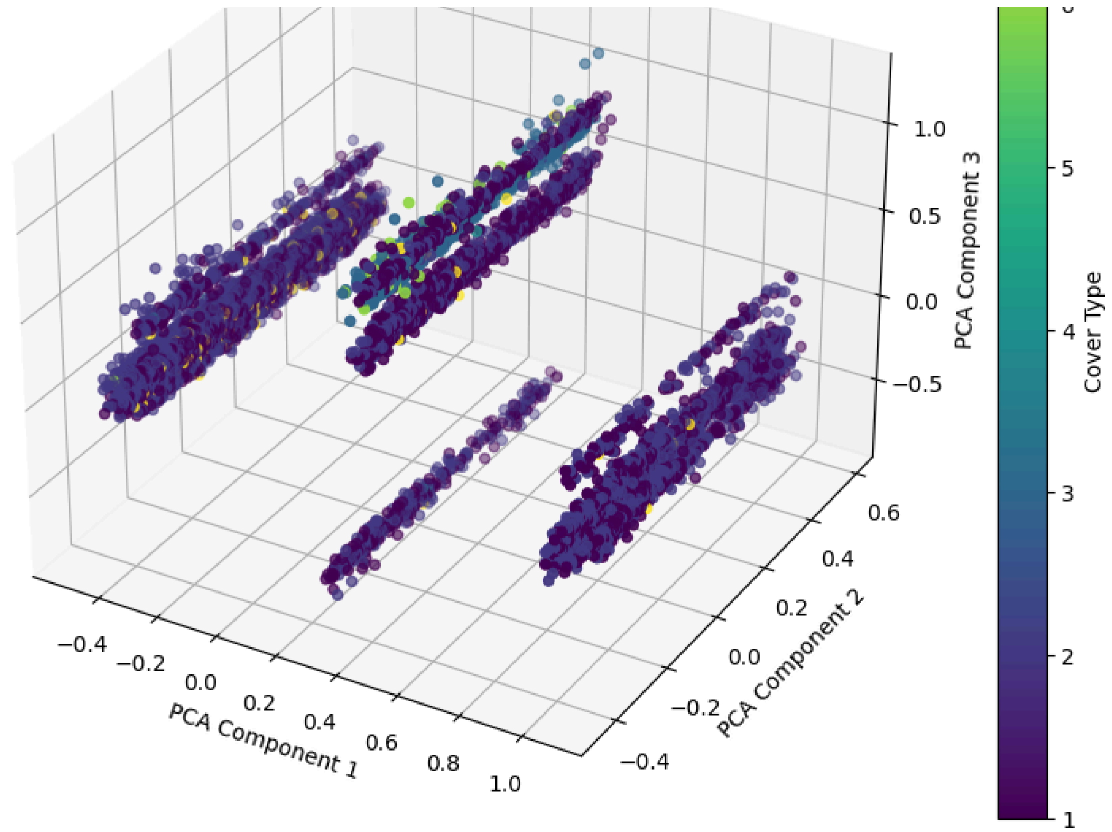
A7 Fill the below cells. Use extra cells as per your necessary

In [21]:
```
df_plot = pd.concat([data_subset_x, data_subset_y.rename('Cover_Type')], axis=1
```

In [22]:
```
# Plotting in 2D
plt.figure(figsize=(10, 6))
plt.scatter(df_plot['PCA_Component_1'], df_plot['PCA_Component_2'], c=df_plot['
plt.title('PCA Scatter Plot (2D)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.colorbar(label='Cover Type')
plt.show()

# Plotting in 3D
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(df_plot['PCA_Component_1'], df_plot['PCA_Component_2'], df
ax.set_title('PCA Scatter Plot (3D)')
ax.set_xlabel('PCA Component 1')
ax.set_ylabel('PCA Component 2')
ax.set_zlabel('PCA Component 3')
plt.colorbar(scatter, label='Cover Type')
plt.show()
```







Q8: Now we will plot all points from our dataframe `df_plot` Using the result from **PCA**

1. Use `pca-one` and `pca-two` column as X and Y axis respectively.
2. Use seaborn scatterplot for plotting the points.

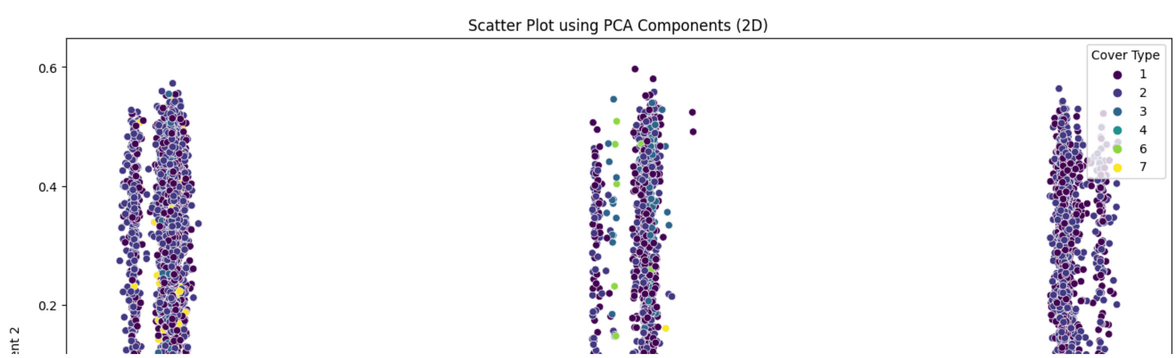
`Note:` Use the notebook from class for reference. The link is provided below.

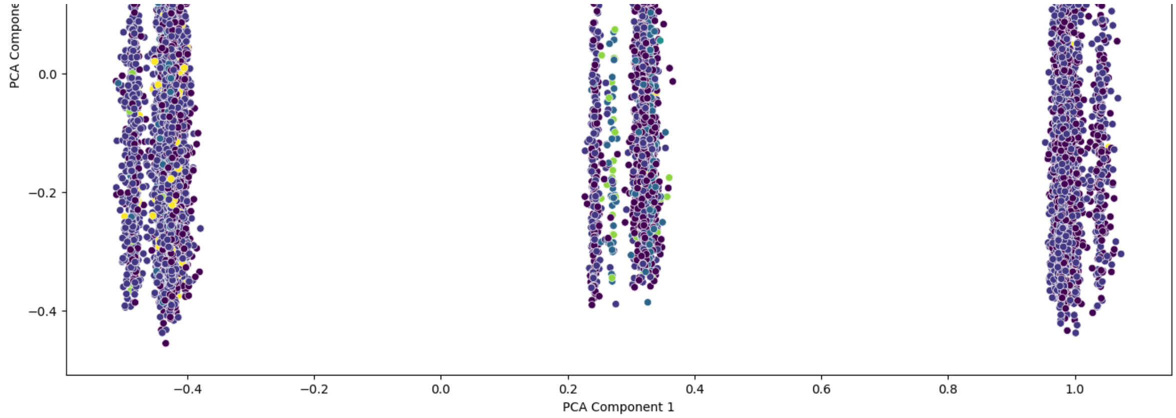
`Link:` https://git.txstate.edu/ML/2024Spring/tree/main/project/examples/Data_Viz_with_PC.

A8 Fill the below cells. Use extra cells as per your necessary

In [23]:

```
plt.figure(figsize=(16, 10))
sns.scatterplot(data=df_plot, x='PCA_Component_1', y='PCA_Component_2', hue='Co
plt.title('Scatter Plot using PCA Components (2D)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(title='Cover Type')
plt.show()
```





Q9: Now we will plot all points from our dataframe `df_plot` Using result from T-SNE.

1. Use `tsne-2d-one` and `tsne-2d-one` column as X and Y axis respectively.
2. Use seaborn scatterplot for plotting the points.

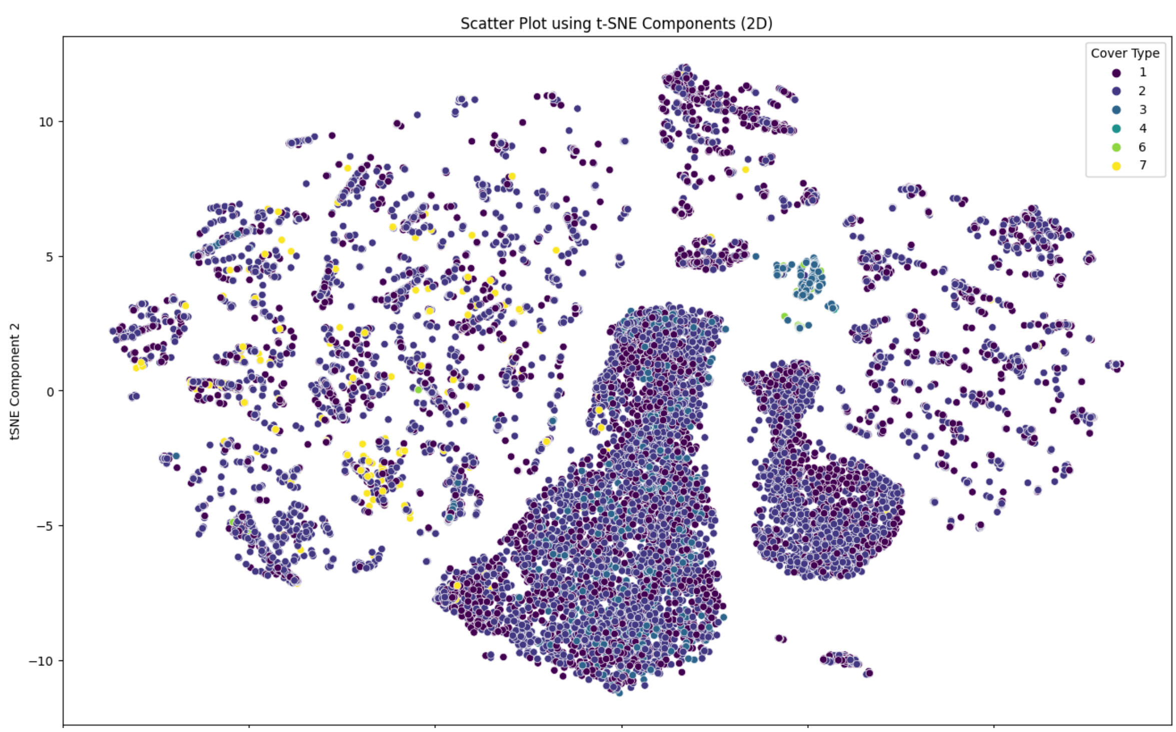
`Note:` Use the notebook from class for reference. The link is provided below.

`Link:`
https://git.txstate.edu/ML/2024Spring/tree/main/project/examples/Data_Viz_with_PCA_TSNE.

A9 Fill the below cells. Use extra cells as per your necessary

In [24]:

```
plt.figure(figsize=(16, 10))
sns.scatterplot(data=df_plot, x='tSNE_Component_1', y='tSNE_Component_2', hue='
plt.title('Scatter Plot using t-SNE Components (2D)')
plt.xlabel('tSNE Component 1')
plt.ylabel('tSNE Component 2')
plt.legend(title='Cover Type')
plt.show()
```



Part 2: Data Analysis and Classification Using Entire Dataset

Q10: Observe the data plotting and find the realtion between datapoints and their characteristics.

1. Reduce the dimension of our `Scaled_X` dataframe to `3` using PCA algorithm.
2. Store the result into a variable named `pca_result`
3. Create Train data and Test data using the pca_result and Y.

`Note:`

1. Consider pca_result as X values, and Y as y values.
2. You can use sklearn train_test_split
3. Keep Train and Test ratio as : 75%:25%

A10 Fill the below cells. Use extra cells as per your necessary

In [25]:
```
#You can create or remove cells as per your need
pca = PCA(n_components=3)
pca_result = pca.fit_transform(Scaled_X)
```

In [26]:
```
x_train, x_test, y_train, y_test = train_test_split(pca_result, Y, test_size=0.
```

Now, Select Three best model for our dataset. You have to decide three models which might work well with our dataset.

Q11

Model Number 1

1. Reason behind choosing the model.
2. Create the model using sklearn or any proper library
3. Fit the model with the train data
4. Get the score from the model using test data

A11 Fill the below cells. Use extra cells as per your necessary

`Answer for Q.No:1 goes here` Random Forest is a versatile and powerful ensemble learning method that is known for its robustness and ability to handle high-dimensional data with complex relationships.

In [27]:
```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import accuracy_score

# Initialize Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42)

# Fit the model with the training data
rf_model.fit(x_train, y_train)

# Get the accuracy score from the model using the test data
rf_score = rf_model.score(x_test, y_test)
print("Accuracy score for Random Forest Classifier:", rf_score)
```

```
Accuracy score for Random Forest Classifier: 0.60825
```

Q12

Model Number 2

1. Reason behind choosing the model.
2. Create the model using sklearn or any proper library
3. Fit the model with the train data
4. Get the score from the model using test data

A12 Fill the below cells. Use extra cells as per your necessaryReplace ??? with code in the code cell below

`Answer for Q.No:1 goes here` SVMs are powerful models known for their effectiveness in high-dimensional spaces, making them suitable for our dataset. They work well with complex relationships and can handle both linear and non-linear data.

In [28]:

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Initialize Support Vector Machine Classifier
svm_model = SVC()

# Fit the model with the training data
svm_model.fit(x_train, y_train)

# Get the accuracy score from the model using the test data
svm_score = svm_model.score(x_test, y_test)
print("Accuracy score for Support Vector Machine Classifier:", svm_score)
```

```
Accuracy score for Support Vector Machine Classifier: 0.5886
```

Q13

Model Number 3

1. Reason behind choosing the model.
2. Create the model using sklearn or any proper library
3. Fit the model with the train data
4. Get the score from the model using test data

A13 Fill the below cells. Use extra cells as per your necessary

`Answer for Q.No:1 goes here` Gradient Boosting is an ensemble learning method that builds multiple decision trees sequentially, where each tree corrects the errors of the previous one. It is known for its high predictive accuracy and ability to handle complex relationships in the data.

In [29]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

# Initialize Gradient Boosting Classifier
gb_model = GradientBoostingClassifier()

# Fit the model with the training data
gb_model.fit(x_train, y_train)

# Get the accuracy score from the model using the test data
gb_score = gb_model.score(x_test, y_test)
print("Accuracy score for Gradient Boosting Classifier:", gb_score)
```

```
Accuracy score for Gradient Boosting Classifier: 0.6114
```

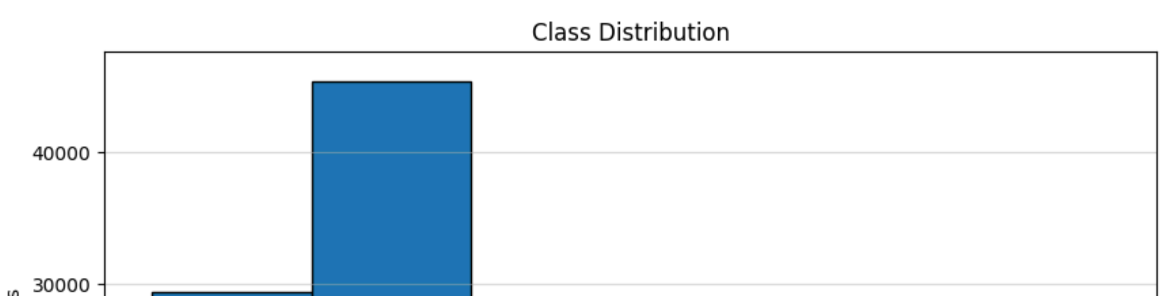
Q14

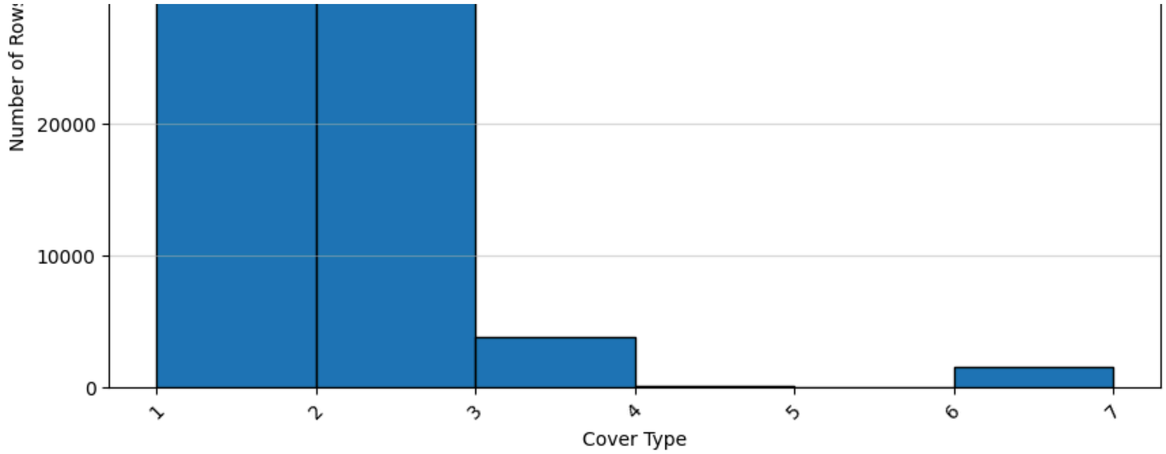
1. Plot a histogram using Y dataframe and display the per-class data distribution(number of rows per class).
2. Also print the number of rows per class as numeric value.

A14 Fill the below cells. Use extra cells as per your necessary

In [30]:

```
import matplotlib.pyplot as plt

# Plot histogram
plt.figure(figsize=(10, 6))
plt.hist(Y, bins=len(Y.unique()), edgecolor='black')
plt.title('Class Distribution')
plt.xlabel('Cover Type')
plt.ylabel('Number of Rows')
plt.xticks(rotation=45)
plt.grid(axis='y', alpha=0.5)
plt.show()

# Print number of rows per class
print("Number of rows per class:")
print(Y.value_counts())
```






```
Number of rows per class:
2    45393
1    29311
3     3816
7     1245
6      229
4        6
Name: Cover_Type, dtype: int64
```

Q15

1. From the histogram we can see that the dataset is highly imbalanced.
2. Use a proper dataset balancing technique to make the dataset balanced.
3. Plot a histogram using new y values and display the per-class data distribution(number of rows per class).

`Note:` Use can use the `imblearn.over_sampling` library for this task. But use appropriate strategy for the method.

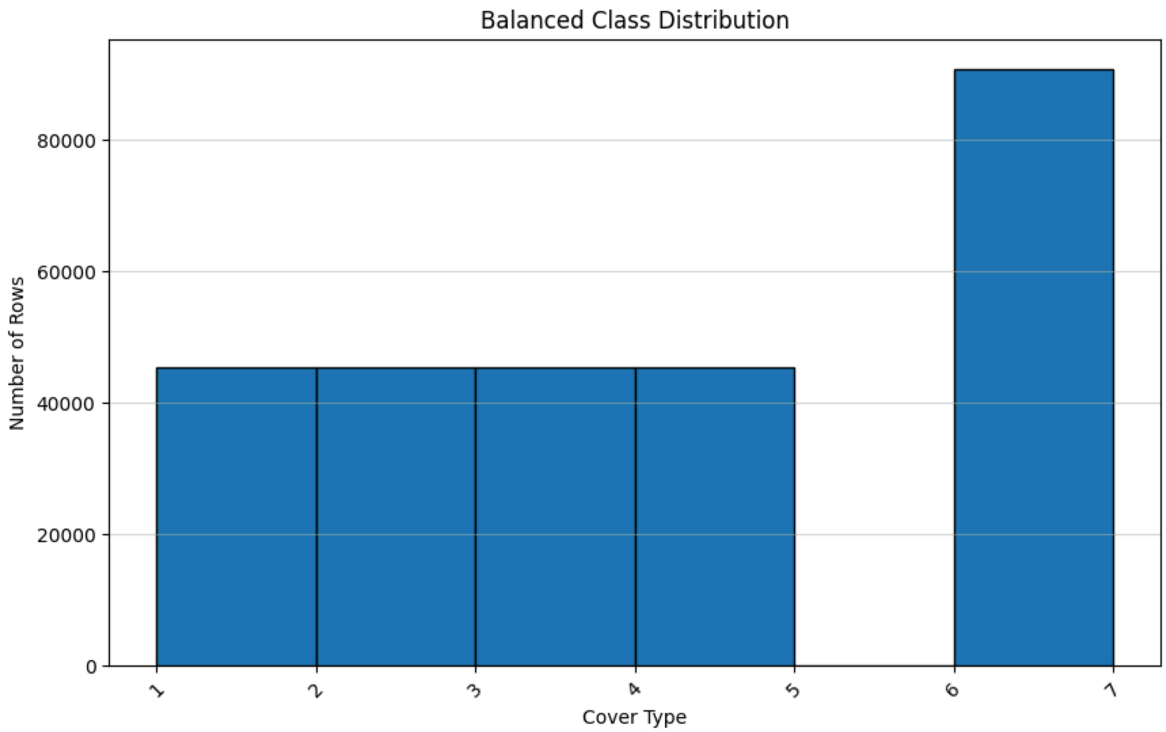
Follow the documentation for details: https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html

A15 Fill the below cells. Use extra cells as per your necessary

In [31]:
```
from imblearn.over_sampling import SMOTE

# Initialize SMOTE
smote = SMOTE(sampling_strategy='auto', random_state=42)

# Resample the dataset
X_res, y_res = smote.fit_resample(Scaled_X, Y)
```

In [32]:
```
import matplotlib.pyplot as plt

# Plot histogram for balanced data
plt.figure(figsize=(10, 6))
plt.hist(y_res, bins=len(y_res.unique()), edgecolor='black')
plt.title('Balanced Class Distribution')
plt.xlabel('Cover Type')
plt.ylabel('Number of Rows')
plt.xticks(rotation=45)
```

```
plt.grid(axis='y', alpha=0.5)
plt.show()
```



Q16

1. Create new Train and Test data from the balaned X and Y value.
2. Keep Train and Test ratio as : 75%:25%

A16 Fill the below cells. Use extra cells as per your necessary

In [33]:
```
from sklearn.model_selection import train_test_split

# Split the balanced data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2
```

Q17

Now, Use the previously initialized three models and calculate the score from our new balanced dataset.

Model Number 1

1. Fit the model with the new train data(Use the previous Model 1)
2. Get the score from the model using new test data

A17 Fill the below cells. Use extra cells as per your necessary

In [34]:
```
# Fit the Random Forest model with the new train data
rf_model.fit(x_train, y_train)
```

```
# Get the accuracy score from the model using the new test data
rf_score_balanced = rf_model.score(x_test, y_test)
print("Accuracy score for Random Forest Classifier with balanced data:", rf_sco
```

```
Accuracy score for Random Forest Classifier with balanced data: 0.982464385372301
4
```

Model Number 2

1. Fit the model with the new train data(Use the previous Model 2)
2. Get the score from the model using new test data

Fill the below cells. Use extra cells as per your necessary

In [35]:

```
# Fit the Support Vector Machine model with the new train data
svm_model.fit(x_train, y_train)

# Get the accuracy score from the model using the new test data
svm_score_balanced = svm_model.score(x_test, y_test)
print("Accuracy score for Support Vector Machine Classifier with balanced data:
```

```
Accuracy score for Support Vector Machine Classifier with balanced data: 0.952195
6234395653
```

Model Number 3

1. Fit the model with the new train data(Use the previous Model 3)
2. Get the score from the model using new test data

Fill the below cells. Use extra cells as per your necessary

In [36]:

```
# Fit the Gradient Boosting model with the new train data
gb_model.fit(x_train, y_train)

# Get the accuracy score from the model using the new test data
gb_score_balanced = gb_model.score(x_test, y_test)
print("Accuracy score for Gradient Boosting Classifier with balanced data:", gb
```

```
Accuracy score for Gradient Boosting Classifier with balanced data: 0.93880158613
59965
```

After making the dataset balanced we can see a significant improve in the performence for all three models.