Individual Report

Outline: -

- **Introduction**
- **Description of your individual work**
- **Describe the portion of the work that you did on the project in detail**
- **Experimental setup**
- **Results**
- **Summary and conclusions**
- **Calculate the percentage of the code that you found or copied from the internet**
- **References**

1. Introduction.

Predicting forest cover types is an important task that can provide valuable information for land managers, conservationists, and scientists who are interested in understanding and managing forest ecosystems. By accurately predicting forest cover types, we can identify areas that are vulnerable to wildfires and other natural disasters, predict the impact of climate change on forest ecosystems, and identify areas that are suitable for timber harvesting or recreational use**.**

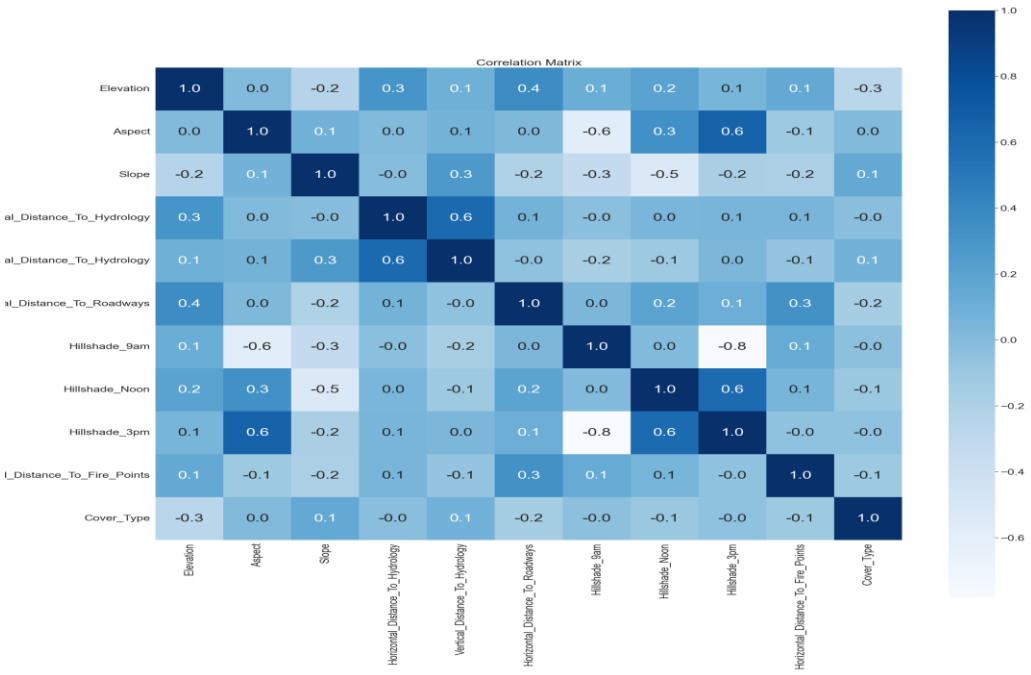
Accurate prediction of forest cover types can provide valuable information for land managers, conservationists, and scientists who are interested in understanding and managing forest ecosystems. For example, accurate predictions of forest cover types can help identify areas vulnerable to wildfires, predict the impact of climate change on forest ecosystems, and identify areas suitable for timber harvesting or recreational use.

Moreover, the dataset is also useful for evaluating the performance of machine learning algorithms and techniques for classification tasks. The dataset is large and contains a diverse range of features, including both quantitative and categorical variables, making it a challenging but realistic problem for machine learning models to solve.

2. Description of your individual work.

EDA:

I wrote the code for the correlation plot using the 10 numeric features to find which feature is highly correlated with the target variable forest cover type. The correlation plot shows us that there are features which are positively and weakly correlated like horizontal distance to hydrology and a few features like elevation that are negatively and weakly correlated with forest cover type (target class).



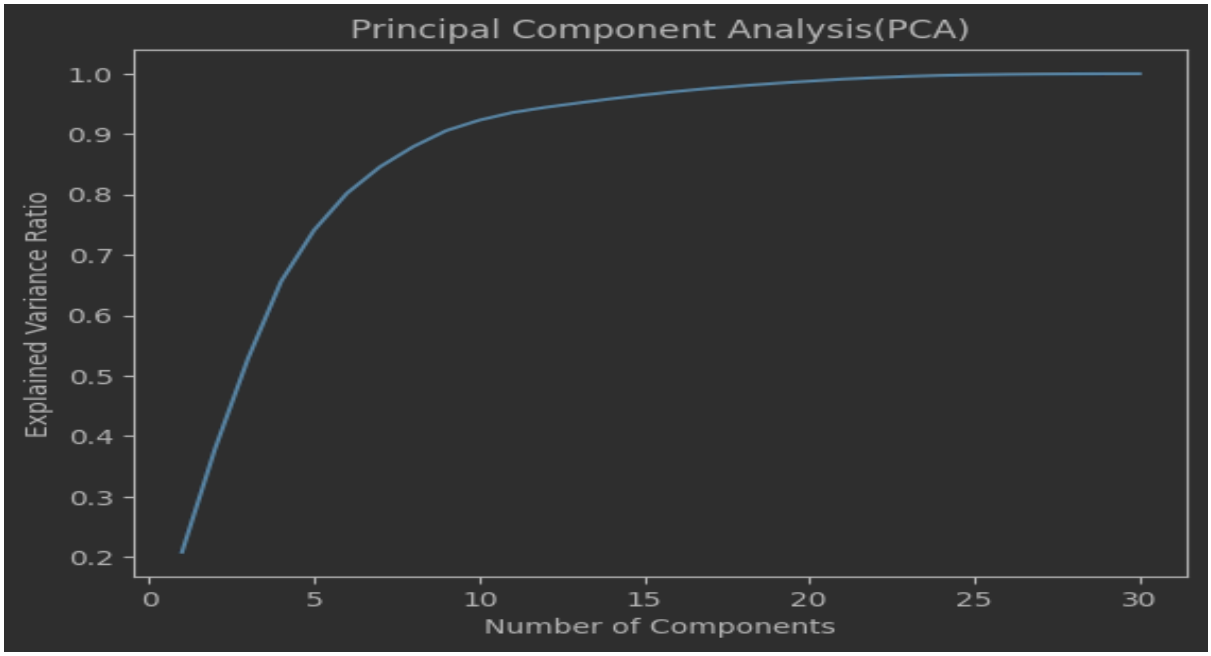
PCA:

For feature reduction after using recursive feature reduction, we came across 30 features that we can use from our dataset. To ensure that we are not losing out on important features, I also did a PCA and checked the explained variance ratio.

The 30 features are: -

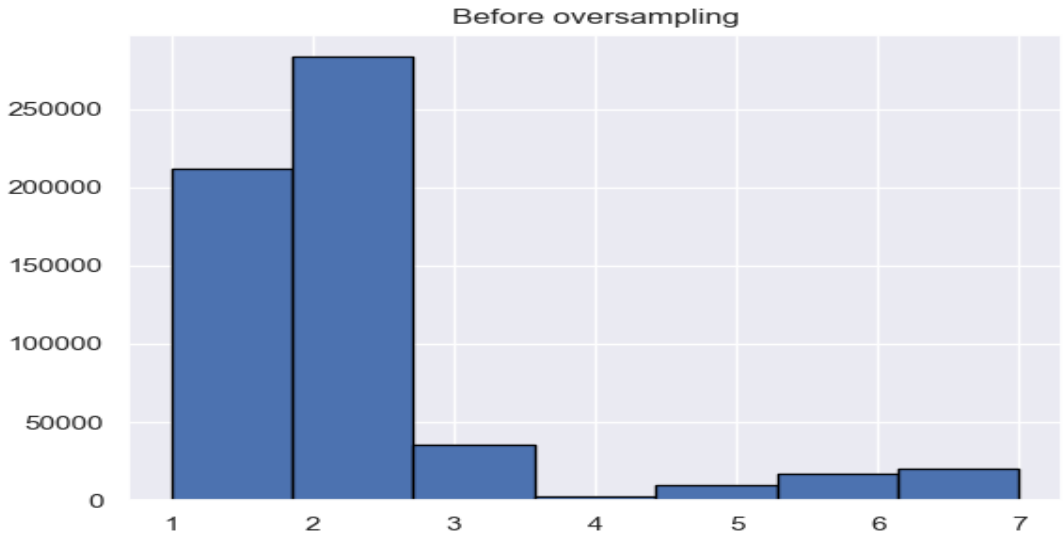
- Elevation
- Slope
- Horizontal_Distance_To_Hydrology
- Vertical_Distance_To_Hydrology
- Horizontal_Distance_To_Roadways
- Hillshade_9am
- Hillshade_Noon
- Hillshade_3pm

- Horizontal_Distance_To_Fire_Points
- Wilderness_Area1
- Wilderness_Area2
- Wilderness_Area3
- Wilderness_Area4
- Soil_Type2
- Soil_Type4
- Soil_Type10
- Soil_Type12
- Soil_Type22
- Soil_Type23
- Soil_Type24
- Soil_Type29
- Soil_Type32
- Soil_Type33
- Soil_Type38
- Inceptisols
- Mollisols
- Spodosols
- Alfisols
- Entisols
- Aspect_unimodal
- Cover_Type



Looking at the above plot we can see that almost 99% of explained variance ratio is explained by these 30 features.

SMOTE (Synthetic Minority Oversampling Technique): -
Since the dataset was imbalanced with most samples falling under classes 1 and 2, we generated synthetic data using SMOTE and balanced the target feature.



MLP classifier: -
I used an MLP classifier algorithm for the multi-class classification problem. I defined the parameter space for the MLP classifier as follows: -

```
parameter_space = {
    'hidden_layer_sizes': [(50,),(100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
```

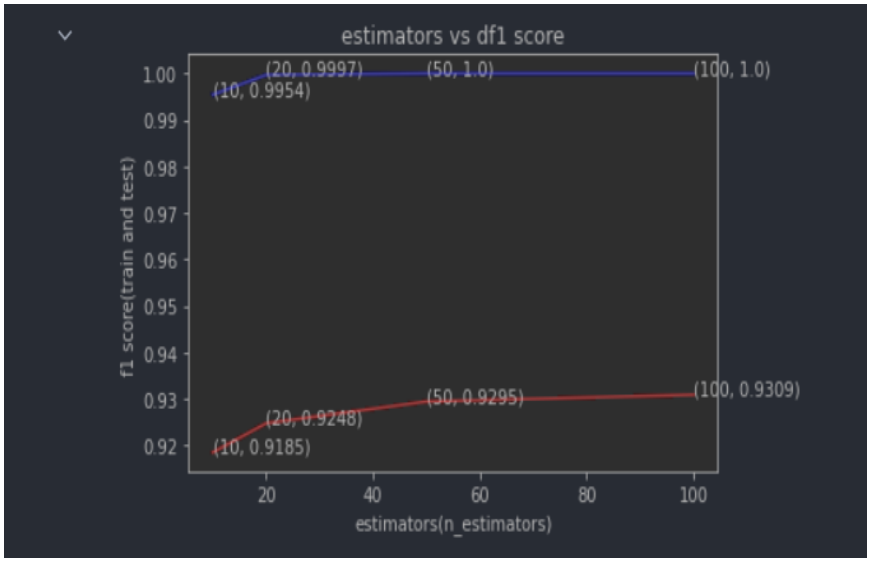
I made use of a Grid search to find the best set of parameters from the parameter space defined above. I found the best set of parameters to be as: -

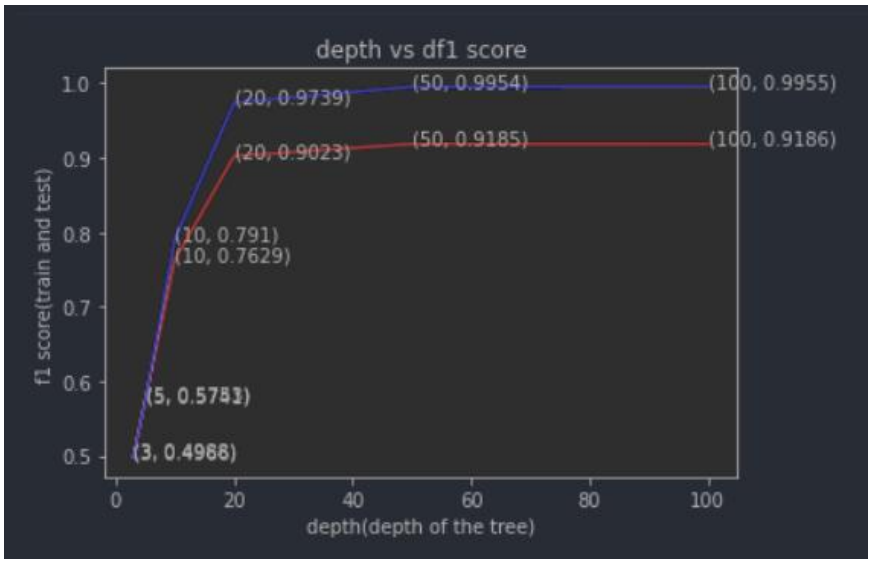
```
clf.best_params_
```

```
{'activation': 'relu',
 'alpha': 0.0001,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'solver': 'adam'}
```

XGBoost classifier

For the XGBoost classifier, I used a similar approach for hyperparameter tuning as the Random Forest classifier by defining some initial values for the parameters number of estimators and the depth.





3. Describe the portion of the work that you did on the project in detail. It can be figures,
codes, explanations, pre-processing, training, etc.
EDA- Correlation plot

```
# Heatmap for correlation
correlation_matrix = df.corr()
k = 55 # number of variables for heatmap
cols = correlation_matrix.nlargest(k,'Cover_Type')['Cover_Type'].index
cm = np.corrcoef(df[cols].values.T)
sns.set(font_scale=2)
fig, ax = plt.subplots(figsize=(30,30))  # Sample figsize in inches
hm = sns.heatmap(cm, cbar=True, annot=True, cmap = "Blues",
                 square=True, fmt='.01f',
                 annot_kws={'size': 20},
                 yticklabels=cols.values,
                 xticklabels=cols.values,ax=ax)
plt.title("Correlation Matrix")
plt.show()
```

PCA (Principal Component Analysis)

```
pca = PCA(n_components=30)
pca.fit(top_30_features)

import matplotlib.pyplot as plt
plt.plot(range(1,len(pca.explained_variance_ratio_)+1), pca.explained_variance_ratio_.cumsum())
plt.title("Principal Component Analysis(PCA)")
plt.xlabel('Number of Components')
plt.ylabel('Explained Variance Ratio')
plt.show()
```

SMOTE (Synthetic Minority Oversampling Technique)

```
from imblearn.over_sampling import SMOTE
```
Executed at 2023.05.02 13:26:30 in 152ms

```
oversample = SMOTE()
X_train_smote, Y_train_smote = oversample.fit_resample(x_train, y_train)
```
Executed at 2023.05.02 13:30:44 in 4m 13s 976ms

```
Y_train_smote.value_counts()
```
Executed at 2023.05.02 13:30:44 in 42ms

7 rows Length: 7, dtype: int64 pd.Series CSV

	Cover_Type
2	158481
1	158481
3	158481
6	158481
5	158481
7	158481
4	158481

MLP classifier

XGBoost classifier

```
from sklearn.metrics import f1_score
from xgboost import XGBClassifier
depth = [3,5,10,20,50,100]
cv_f1_score = []
train_f1_score = []
for i in depth:
  model = XGBClassifier(max_depth = i,n_jobs = -1,n_estimators = 10)
  model.fit(X_train_smote, Y_train_smote)
  CV = CalibratedClassifierCV(model,method = 'sigmoid')
  CV.fit(X_train_smote, Y_train_smote)
  predicted = CV.predict(x_cv)
  train_predicted = CV.predict(x_train)
  cv_f1_score.append(f1_score(y_cv,predicted,average = 'macro'))
  train_f1_score.append(f1_score(y_train,train_predicted,average = 'macro'))
  print('depth {0} is finished'.format(i))
for i in range(0,len(cv_f1_score)):
  print('f1 value score for depth =' + str(depth[i]) + ' is ' + str(cv_f1_score[i]))
plt.plot(depth,cv_f1_score,c='r')
plt.plot(depth,train_f1_score,c='b')
plt.xlabel('depth(depth of the tree)')
plt.ylabel('f1 score(train and test)')
plt.title('depth vs df1 score')
```

```
plt.title('depth vs df1 score')
for i,score in enumerate(cv_f1_score):
  plt.annotate((depth[i],np.round(score,4)),(depth[i],np.round(cv_f1_score[i],4)))
for i,score1 in enumerate(train_f1_score):
  plt.annotate((depth[i],np.round(score1,4)),(depth[i],np.round(train_f1_score[i],4)))
index = cv_f1_score.index(max(cv_f1_score))
best_dpt = depth[index]
print('best max depth is ' + str(best_dpt))
model = XGBClassifier(max_depth = best_dpt,n_jobs = -1)
model.fit(X_train_smote, Y_train_smote)
predict_train = model.predict(x_train)
print('f1 score on train data ' + str(f1_score(y_train,predict_train,average = 'macro')))
train_mat = confusion_matrix(y_train,predict_train)
predict_cv = model.predict(x_cv)
print('f1 score on cv data ' + str(f1_score(y_cv,predict_cv,average = 'macro')))
cv_mat = confusion_matrix(y_cv,predict_cv)
predict_test = model.predict(x_test)
print('f1 score on test data ' + str(f1_score(y_test,predict_test,average = 'macro')))
test_mat = confusion_matrix(y_test,predict_test)
fig,ax = plt.subplots(1,3,figsize = (15,5))
sns.heatmap(ax = ax[0],data = train_mat,annot=True,fmt='g',cmap="YlGnBu")
ax[0].set_xlabel('predicted')
```

```
sns.heatmap(ax = ax[1],data = cv_mat,annot=True,fmt='g')
ax[1].set_xlabel('predicted')
ax[1].set_ylabel('actual')
ax[1].title.set_text('confusion matrix for CV data')
sns.heatmap(ax = ax[2],data = test_mat,annot=True,fmt='g')
ax[2].set_xlabel('predicted')
ax[2].set_ylabel('actual')
ax[2].title.set_text('confusion matrix for test data')
```

Executed at 2023.05.02 15:51:55 in 1h 1m 39s

```
depth 3 is finished
depth 5 is finished
depth 10 is finished
depth 20 is finished
depth 50 is finished
depth 100 is finished
f1 value score for depth =3 is 0.4996588354020641
f1 value score for depth =5 is 0.5778357373870923
f1 value score for depth =10 is 0.759795061645674
f1 value score for depth =20 is 0.9033005721746704
f1 value score for depth =50 is 0.920038402342427
f1 value score for depth =100 is 0.9201263103354325
```

```
depth 50 is finished
depth 100 is finished
f1 value score for depth =3 is 0.4996588354020641
f1 value score for depth =5 is 0.5778357373870923
f1 value score for depth =10 is 0.759795061645674
f1 value score for depth =20 is 0.9033005721746704
f1 value score for depth =50 is 0.920038402342427
f1 value score for depth =100 is 0.9201263103354325
best max depth is 100
f1 score on train data 1.0
f1 score on cv data 0.9330045660999421
f1 score on test data 0.9356208144474581
```

```
fig,ax = plt.subplots(1,3,figsize = (15,5))
sns.heatmap(ax = ax[0],data = train_mat,annot=True,fmt='g',cmap="Blues")
ax[0].set_xlabel('predicted')
ax[0].set_ylabel('actual')
ax[0].title.set_text('confusion matrix for train data')
sns.heatmap(ax = ax[1],data = cv_mat,annot=True,fmt='g',cmap="Blues")
ax[1].set_xlabel('predicted')
ax[1].set_ylabel('actual')
ax[1].title.set_text('confusion matrix for CV data')
sns.heatmap(ax = ax[2],data = test_mat,annot=True,fmt='g',cmap="Blues")
ax[2].set_xlabel('predicted')
ax[2].set_ylabel('actual')
ax[2].title.set_text('confusion matrix for test data')
```

```
from sklearn.metrics import f1_score
estimators = [10,20,50,100]
cv_f1_score = []
train_f1_score = []
for i in estimators:
  model = XGBClassifier(n_estimators = i,max_depth = 50,n_jobs = -1)
  model.fit(X_train_smote, Y_train_smote)
  CV = CalibratedClassifierCV(model,method = 'sigmoid')
  CV.fit(X_train_smote, Y_train_smote)
  predicted = CV.predict(x_cv)
  train_predicted = CV.predict(x_train)
  cv_f1_score.append(f1_score(y_cv,predicted,average = 'macro'))
  train_f1_score.append(f1_score(y_train,train_predicted,average = 'macro'))
  print('Estimator {0} is finished'.format(i))
for i in range(0,len(cv_f1_score)):
  print('f1 value score n_estimators =' + str(estimators[i]) + ' is ' + str(cv_f1_score[i]))
plt.plot(estimators,cv_f1_score,c='r')
plt.plot(estimators,train_f1_score,c='b')
plt.xlabel('estimators(n_estimators)')
plt.ylabel('f1 score(train and test)')
plt.title('estimators vs df1 score')
```

```
for i,score in enumerate(cv_f1_score):
  plt.annotate((estimators[i],np.round(score,4)),(estimators[i],np.round(cv_f1_score[i],4)))
for i,score1 in enumerate(train_f1_score):
  plt.annotate((estimators[i],np.round(score1,4)),(estimators[i],np.round(train_f1_score[i],4)))
index = cv_f1_score.index(max(cv_f1_score))
best_est = estimators[index]
print('best estimator is ' + str(best_est))
model = XGBClassifier(n_estimators = best_est,max_depth = 50,n_jobs = -1)
model.fit(X_train_smote, Y_train_smote)
predict_train = model.predict(X_train_smote)
print('f1 score on train data ' + str(f1_score(Y_train_smote,predict_train,average = 'macro')))
train_mat = confusion_matrix(Y_train_smote,predict_train)
predict_cv = model.predict(x_cv)
print('f1 score on cv data ' + str(f1_score(y_cv,predict_cv,average = 'macro')))
cv_mat = confusion_matrix(y_cv,predict_cv)
predict_test = model.predict(x_test)
print('f1 score on test data ' + str(f1_score(y_test,predict_test,average = 'macro')))
test_mat = confusion_matrix(y_test,predict_test)
fig,ax = plt.subplots(1,3,figsize = (15,5))
sns.heatmap(ax = ax[0],data = train_mat,annot=True,fmt='g',cmap="YlGnBu")
ax[0].set_xlabel('predicted')
ax[0].set_ylabel('actual')
```

```
ax[0].title.set_text('confusion matrix for train data')
sns.heatmap(ax=ax[1], data=cv_mat, annot=True, fmt='g')
ax[1].set_xlabel('predicted')
ax[1].set_ylabel('actual')
ax[1].title.set_text('confusion matrix for CV data')
sns.heatmap(ax=ax[2], data=test_mat, annot=True, fmt='g')
ax[2].set_xlabel('predicted')
ax[2].set_ylabel('actual')
ax[2].title.set_text('confusion matrix for test data')
```

XGBoost classifier

```
best_xgb_model = XGBClassifier(n_estimators=200,criterion='gini', max_depth=50,max_features = 'auto')
best_xgb_model.fit(X_train_smote, Y_train_smote)
pred=best_xgb_model.predict(x_test)
print(classification_report(y_test, pred))
pred_cv=best_xgb_model.predict(x_cv)
print(classification_report(y_cv, pred_cv))
```

```
mlp_gs = MLPClassifier(max_iter=5)
parameter_space = {
    'hidden_layer_sizes': [(50,),(100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
clf = GridSearchCV(mlp_gs, parameter_space, n_jobs=-1, cv=5)
clf.fit(X_train_smote, Y_train_smote) # X is train samples and y is the corresponding labels

print(clf.best_params_)
```

```
mlp_gs = MLPClassifier(activation= 'relu',
alpha= 0.0001,
hidden_layer_sizes= (100,),
learning_rate= 'constant',
solver= 'adam')
mlp_gs.fit(X_train_smote, Y_train_smote)

pred=mlp_gs.predict(x_test)
print(classification_report(y_test, pred))
pred_cv=mlp_gs.predict(x_cv)
print(classification_report(y_cv, pred_cv))
```

4. Results.

XGBoost Classifier turned out to be the best model in terms of F1 score. The reason why MLP Classifier did not perform better than other models is because the dataset used here consists of heterogeneous columns or features which are treated independently, whereas MLP Classifier works best for homogeneous data i.e., pixels in an image, frames in a video, words in a text. Both XGBoost and Random Forest are boosting and bagging models in machine learning where in the case of boosting you combine many models sequentially to improve model predictions while in bagging you combine many models parallelly and determine the best model by voting.

5. Summary and conclusions

- Feature engineering and hyperparameter tuning improved results for the Random Forest Classifier and XGBoost Classifier by 1% and 1.2% respectively.
- Soil order type features were also considered important features in feature selection.
- Some other features like weather conditions and geographical information like location could help us in improving our results further.

6. Calculate the percentage of the code that you found or copied from the internet.

Link from where copied from: - https://amueller.github.io/aml/04-model-evaluation/11-calibration.html

Lines copied: -

```
clf = RandomForestClassifier(n_estimators=25)
clf.fit(X_train, y_train)
clf_probs = clf.predict_proba(X_test)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid", cv="prefit")
sig_clf.fit(X_valid, y_valid)
```

```
sig_clf_probs = sig_clf.predict_proba(X_test)
```

Lines modified: -

```
# Plot changes in predicted probabilities via arrows
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
colors = ["r", "g", "b"]
for i in range(clf_probs.shape[0]):
    plt.arrow(clf_probs[i, 0], clf_probs[i, 1],
              sig_clf_probs[i, 0] - clf_probs[i, 0],
              sig_clf_probs[i, 1] - clf_probs[i, 1],
              color=colors[y_test[i]], head_width=1e-2)
```

Lines added: -

```
CV.fit(X_train_smote, Y_train_smote)
       predicted = CV.predict(x_cv)
       train_predicted = CV.predict(x_train)
       cv_f1_score.append(f1_score(y_cv, predicted, average='macro'))
       train_f1_score.append(f1_score(y_train, train_predicted, average='macro'))
       print('depth {0} is finished'.format(i))
   for i in range(0, len(cv_f1_score)):
       print('f1 value score for depth =' + str(depth[i]) + ' is ' +
str(cv_f1_score[i]))
   plt.plot(depth, cv_f1_score, c='r')
   plt.plot(depth, train_f1_score, c='b')
   plt.xlabel('depth(depth of the tree)')
   plt.ylabel('f1 score(train and test)')
   plt.title('depth vs df1 score')
   for i, score in enumerate(cv_f1_score):
       plt.annotate((depth[i], np.round(score, 4)), (depth[i],
np.round(cv_f1_score[i], 4)))
   for i, score1 in enumerate(train_f1_score):
       plt.annotate((depth[i], np.round(score1, 4)), (depth[i],
np.round(train_f1_score[i], 4)))
   index = cv_f1_score.index(max(cv_f1_score))
   best_dpt = depth[index]
   print('best max depth is ' + str(best_dpt))
   model = XGBClassifier(max_depth=best_dpt, n_jobs=-1)
   model.fit(X_train_smote, Y_train_smote)
   predict_train = model.predict(x_train)
```

```
    print('f1 score on train data ' + str(f1_score(y_train, predict_train,
average='macro')))
    train_mat = confusion_matrix(y_train, predict_train)
    predict_cv = model.predict(x_cv)
    print('f1 score on cv data ' + str(f1_score(y_cv, predict_cv,
average='macro')))
    cv_mat = confusion_matrix(y_cv, predict_cv)
    predict_test = model.predict(x_test)
    print('f1 score on test data ' + str(f1_score(y_test, predict_test,
average='macro')))
```

% of lines found or copied = 6 – 8 / 30 = 5.74%

7. References.

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html

https://seaborn.pydata.org/examples/many_pairwise_correlations.html