


**Name:** Mohamed Hussein Fathy Mohamed Hassan

**Neptun:** FMFYDN

#	Δpub	Team Name	Notebook	Team Members	Score 🏆	Entries	Last
1	▲ 2	Mohamed Hussein			0.97650	2	2d

## ▼ Forest Type Predication:

Content:

- Data Statistics and Understanding
- Data Visualization
- Data Preparation and Cleaning
- Feature selection
- Predaction Model Traning
- Evaluation and Analysis

## ▼ Data Statistics and Understanding:

**Step1:** Check the data Measurement levels:

we need to check the data Measurement levels (Categorical, Numerical)

Code:

```
# check the data Measuremment levels
train_x.head()
# double check using the dtypes method
train_x.dtypes
```

Numerical:

1. Elevation
2. Aspect
3. Slope
4. Horizontal\_Distance\_To\_Hydrology
5. Vertical\_Distance\_To\_Hydrology
6. Horizontal\_Distance\_To\_Roadways
7. Hillshade\_9am
8. Hillshade\_Noon
9. Hillshade\_3pm

- from feature 1~9 are already Numerical

Categorical:

1. Wilderness\_Area
2. Soil types

- from feature 10~51 were Categorical, but It already applied one-hot encoder to them so they are ready to use

	id	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways	Hillshade_9am
0	0	3342	15	13	350	55	2118	208
1	1	2764	39	13	175	43	2648	220
2	2	2773	158	5	162	14	2012	226
3	3	3083	108	16	30	-14	2639	246
4	4	3096	40	4	201	38	4592	220

5 rows × 55 columns

**Conclusion:** overall The dataset is Numerical and ready to use.

## Step2: Check Missing Data:

we need to check the Missing values in the dataset.

Code:

```
# Check Missing Data
train_x.info(null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 406708 entries, 0 to 406707
Data columns (total 55 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   id                                         406708 non-null  int64
1   Elevation                                 406708 non-null  int64
2   Aspect                                   406708 non-null  int64
3   Slope                                    406708 non-null  int64
4   Horizontal_Distance_To_Hydrology          406708 non-null  int64
5   Vertical_Distance_To_Hydrology            406708 non-null  int64
6   Horizontal_Distance_To_Roadways           406708 non-null  int64
7   Hillshade_9am                             406708 non-null  int64
8   Hillshade_Noon                           406708 non-null  int64
9   Hillshade_3pm                            406708 non-null  int64
10  Horizontal_Distance_To_Fire_Points         406708 non-null  int64
```

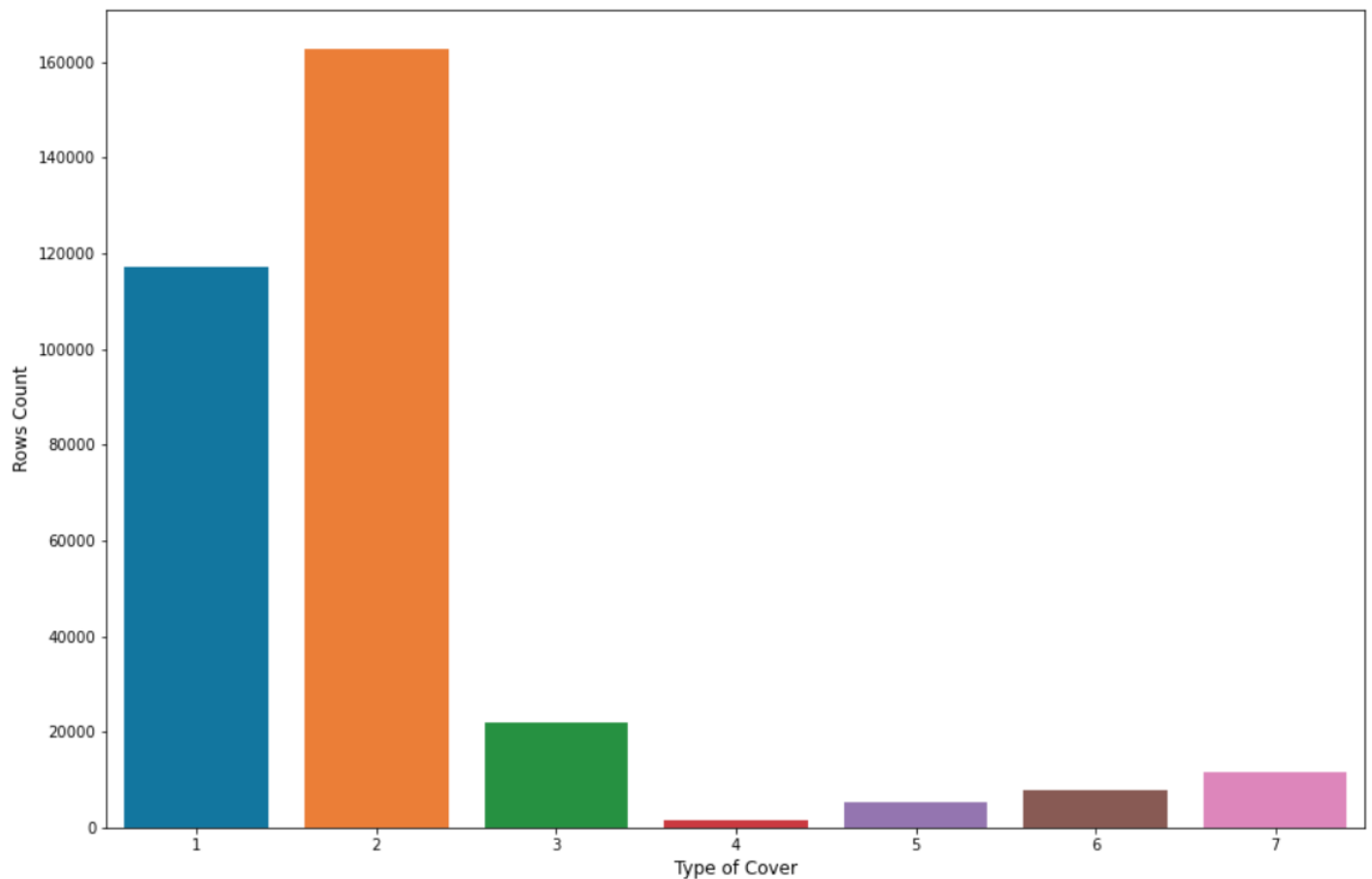
**Conclusion:** There are no missing values in the dataset.

### Step3: Check if Data balance:

check the the balance between the predication classes. we can do that by plotting histogram between the Cover types classes

Code:

```
plt.figure(figsize=(15,10))
sns.countplot(train_y['Cover_Type'])
plt.xlabel("Type of Cover", fontsize=12)
plt.ylabel("Rows Count", fontsize=12)
plt.show()
```



**Conclusion:** The dataset are imbalance and most of the data are for Type Cover (1, 2).

# Data Visualization

## Step1: Check Outliers:

check outliers for numerical features:

1. Elevation
2. Aspect
3. Slope
4. Horizontal\_Distance\_To\_Hydrology
5. Vertical\_Distance\_To\_Hydrology
6. Horizontal\_Distance\_To\_Roadways
7. Hillshade\_9am
8. Hillshade\_Noon
9. Hillshade\_3pm

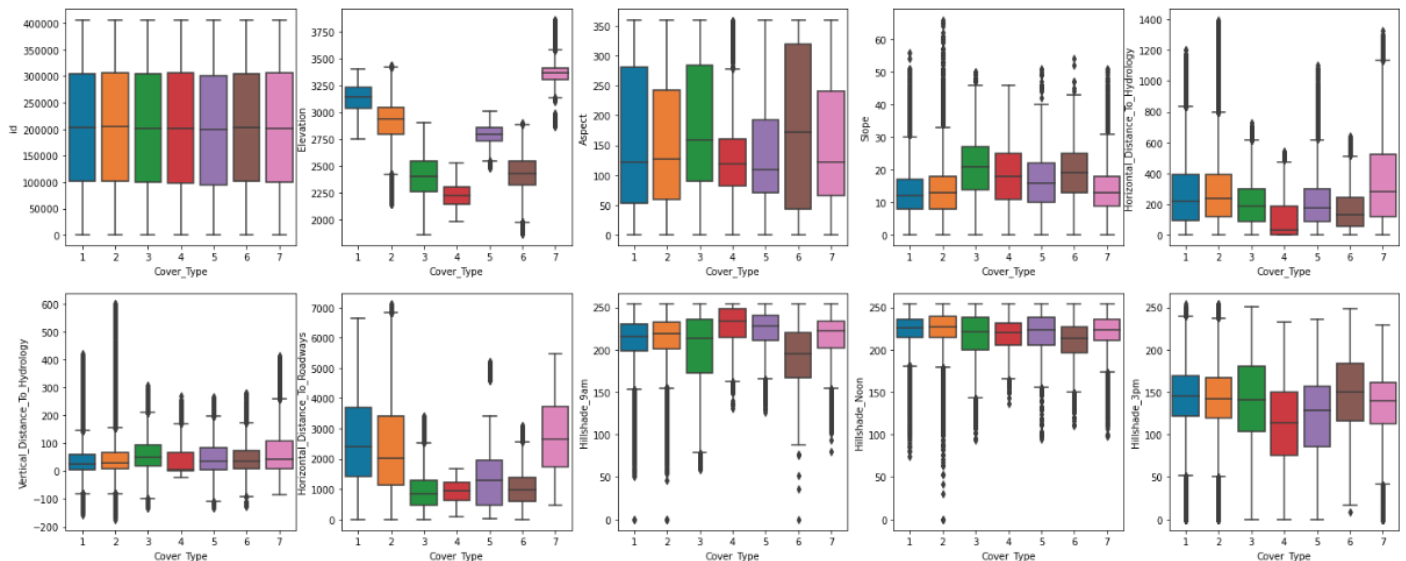
and for each feature check the outliers for each Cover Type.

we can do that but plotting box plot which is a method for graphically depicting groups of numerical data through their quartiles. and Outliers plotted as individual points.

Code:

```
size=10
fig, axes = plt.subplots(nrows = 2,ncols = 5,figsize = (25,10))
for i in range(0,size):
    row = i // 5
    col = i % 5
    ax_curr = axes[row, col]
    sns.boxplot(x="Cover_Type", y=data.iloc[:,i], data=data,ax=ax_curr);
```

Repeat this step for each feature and detect the outliers



**Conclusion:** Yes there are a lot of outliers. we will handle them in the data cleaning section.

## ▼ Data Preparation and Cleaning

### Step1: Remove Outliers:

in this step, we will remove the outliers from features (1 to 9). we already detect the box plot minimum and maximum limit from the previous visualization now we need to get rid of these outliers.

this sample code for removing the outliers from Elevation features Cover\_Type(2,5,6,7). Code:

```
trees = trees[~(((traindata['Elevation'] < 2500) |  
                (traindata['Elevation'] > 3300)) &  
                (traindata['Cover_Type'] == 2))]  
trees = trees[~(((traindata['Elevation'] < 2550) |  
                (traindata['Elevation'] > 3000)) &  
                (traindata['Cover_Type'] == 5))]  
trees = trees[~(((traindata['Elevation'] < 2100) |  
                (traindata['Elevation'] > 2800)) &  
                (traindata['Cover_Type'] == 6))]  
trees = trees[~(((traindata['Elevation'] < 3200) |  
                (traindata['Elevation'] > 3550)) &  
                (traindata['Cover_Type'] == 7))]
```

Repeat this step for each feature and remove the outliers

**Conclusion:** now we have clean data from most of the outliers

---

## Data Preparation and Cleaning

### Step2: Data Standardization:

**hint1:** This step is not mandatory, but just in case I make use of linear models.

**hint2:** This step is for (Train and Test) data.

From checking the Numerical data features ( 1 to 9), we can notice a huge variation range between the features.

Code:

```
# from sklearn.preprocessing import StandardScaler  
# sc = StandardScaler()  
# f1 = 0  
# f9 = 9
```

```
# sc.fit_transform(train_x.iloc[:,f1:f9], inplace=True)
# sc.fit_transform(test_x.iloc[:,f1:f9], inplace=True)
```

**Conclusion:** now we have Standardized data which means that mean equal to zero and std equal to 1

---

## ▼ Feature selection

**Step1:** Remove the features columns with very small standard deviation:

**hint1:** This step is for (Train and Test) data.

the features which standard deviation are zeros or almost zeros that is mean almost all the feature data are equal and there is no information gain from this feature. So, remove this feature will be better for the model to train faster and train on informative data.

Code:

```
train_x.std()
# remove These features from Train data
train_x.drop(['Soil_Type_7'], axis=1, inplace=True)
train_x.drop(['Soil_Type_8'], axis=1, inplace=True)
train_x.drop(['Soil_Type_15'], axis=1, inplace=True)
train_x.drop(['Soil_Type_36'], axis=1, inplace=True)
train_x.drop(['Soil_Type_37'], axis=1, inplace=True)

# remove These features from Test data
test_x.drop(['Soil_Type_7'], axis=1, inplace=True)
test_x.drop(['Soil_Type_8'], axis=1, inplace=True)
test_x.drop(['Soil_Type_15'], axis=1, inplace=True)
test_x.drop(['Soil_Type_36'], axis=1, inplace=True)
test_x.drop(['Soil_Type_37'], axis=1, inplace=True)
```

Example: **Soil Type 15**

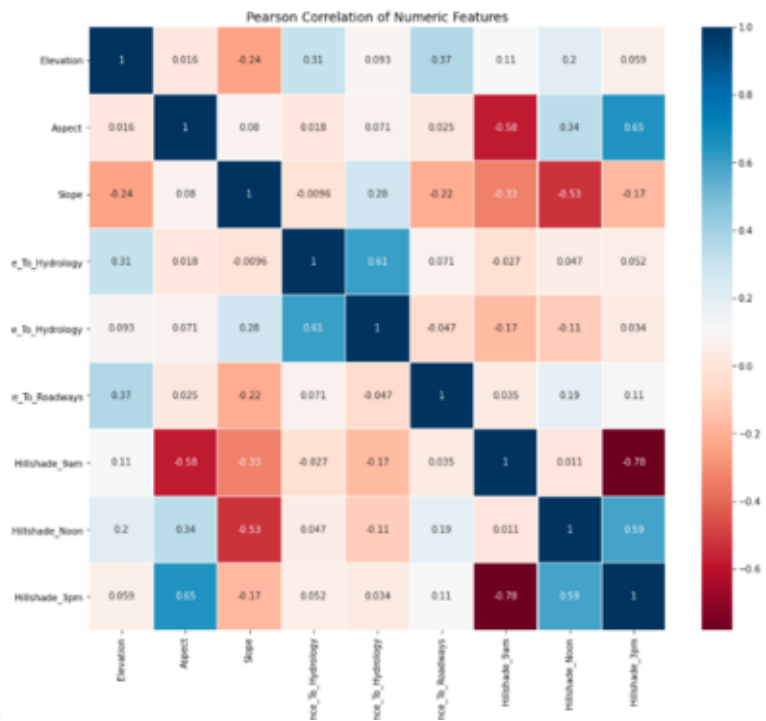
**0.002218**

**Conclusion:** Soil\_Type\_7, Soil\_Type\_8, Soil\_Type\_15, Soil\_Type\_36, Soil\_Type\_37 have a very small standard deviation so we dropped them from the dataset(train, test)

---

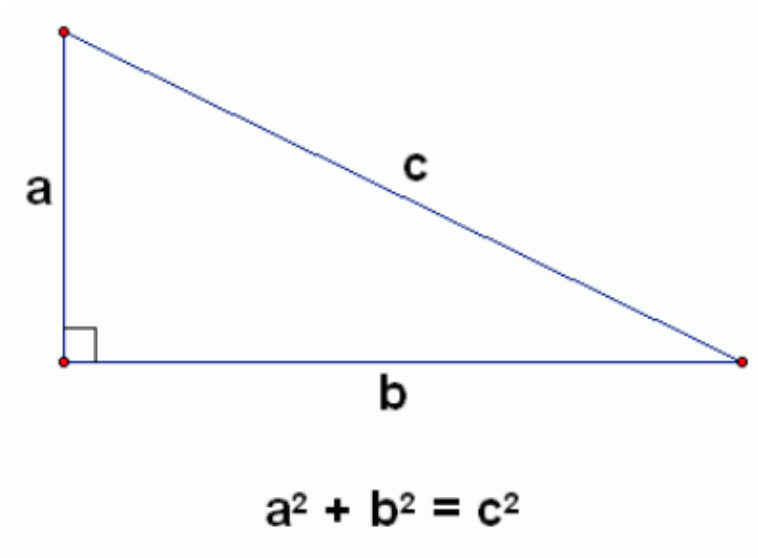
**Step2:** Data Correlation:

**hint1:** This step is for (Train and Test) data.



we can see the data correlation matrix.

### 1. Horizontal\_Distance\_To\_Hydrology and Vertical\_Distance\_To\_Hydrology



There is a relation between Horizontal\_Distance\_To\_Hydrology & Vertical\_Distance\_To\_Hydrology, which make sense because we can replace them with feature represent the diagonal between them using the pythagoras theorem.

Code:

```
train_x.std()
# add new feature
train_x['Diagonal_Distance_to_Hydrology'] = (train_x['Horizontal_Distance_To_Hydrology']**2 + train_x['Vertical_Distance_To_Hydrology']**2)**0.5

train_x.drop(['Horizontal_Distance_To_Hydrology'], axis=1, inplace=True)
train_x.drop(['Vertical_Distance_To_Hydrology'], axis=1, inplace=True)
```

```
# remove These features from Test data
test_x['Diagonal_Distance_to_Hydrology'] = (test_x['Horizontal_Distance_To_Hydrology']**2
+test_x['Vertical_Distance_To_Hydrology']**2)**0.5

test_x.drop(['Horizontal_Distance_To_Hydrology'], axis=1, inplace=True)
test_x.drop(['Vertical_Distance_To_Hydrology'], axis=1, inplace=True)
```

**Conclusion:** There is a relation between Horizontal\_Distance\_To\_Hydrology & Vertical\_Distance\_To\_Hydrology

---

## Predaction Model Traning:

**Step1:** split the train data to (train and test):

Split the dataset to train and test: with ratio x and y that we had previously defined

- train size = 80%
- test size = 20%
- random\_state for selection with seed 10

Code:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(train_x, train_y['Cover_Type'], test_size =

x_train.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)
x_test.reset_index(drop=True, inplace=True)
y_test.reset_index(drop=True, inplace=True)
```

**Conclusion:** now we have train and test data ready for training the model

---

**Step2:** Choosing the training model:

After evaluation 3 models(**KNN, Random Forest, ExtraTreesClassifier**) on small amount of the data set I got the best accuarcy from ExtraTreesClassifier.

What is ExtraTreesClassifier?



ExtraTreesClassifier is an **ensemble learning** method fundamentally based on decision trees.

ExtraTreesClassifier, like RandomForest, randomizes certain decisions and subsets of data to minimize over-learning from the data and overfitting.

## What is difference between Decision Tree vs Random Forest Extra Trees?

- **Decision Tree (High Variance)** A single decision tree is usually overfits the data it is learning from because it learn from only one pathway of decisions. Predictions from a single decision tree usually don't make accurate predictions on new data.
- **Random Forest (Medium Variance)** Random forest models reduce the risk of overfitting by introducing randomness by:
  - *building multiple trees ( $n_{estimators}$ )\**
  - *drawing observations with replacement (i.e., a bootstrapped sample) splitting nodes on the best split among a random subset of the features selected at every node.\**
- **Extra Trees (Low Variance)** Extra Trees is like Random Forest, in that it builds multiple trees and splits nodes using random subsets of features, but with two key differences: it does not bootstrap observations (meaning it samples without replacement), and nodes are split on random splits, not best splits. So, in summary, ExtraTrees:
  - *builds multiple trees with bootstrap = False by default, which means it samples without replacement\**
  - *nodes are split based on random splits among a random subset of the features selected at every node.\**

**Conclusion:** ExtraTreesClassifier make the best predication accuracy.

## Traning Phase

### Step3: Traing Phase:

```
ExtraTreesClassifier(n_estimators=500, criterion = 'entropy',class_weight='balanced')
```

### HyperParamters:

- `criterion = 'entropy':`

measure **quality of a split** of the tree depend on the **information gain** which is calculated from the Entropy.

- `class_weight = 'balanced':`

*\*As we metioned before the dataset is imbalanced \*.so, this parameter "balanced" mode uses the values of Cover Types classes to automatically adjust weights inversely proportional to class frequencies in the input data*

- n\_estimators=500:

this value came from trying [300,400,500] and I compared the results so 500 was the best.

Code:

```
from sklearn.ensemble import ExtraTreesClassifier
train_x.std()
# select the model
treeModel = ExtraTreesClassifier(n_estimators=500, criterion = 'entropy',class_weight='balanced')
# fit the model and start the training
treeModel.fit(x_train, y_train)
```

**Conclusion:** Trained ExtraTreesClassifier model on 80% of the dataset

## **\*\* Evaluation and Analysis\*\***

**Step1:** Measure the Model Accuracy on the test data:

Measure the Model Accuracy on the test data using the accuracy\_score method from sklearn which compare the result between the predicated and the ground truth

Code:

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred)*100)
```

**Conclusion:** accuracy score = 97.39 %