# EARTHQUAKE DAMAGE PREDICTION SYSTEM

PRESENTED BY

ANISH SHILPAKAR

ANUSHIL TIMSINA

SUGAM KARKI

AAROSH DAHAL

# AGENDAS

- DATASET
- EDA
- DATA PREPROCESSING
- MODEL DEVELOPMENT
- MODEL EVALUATION
- MODEL OPTIMIZATION

# DATASET

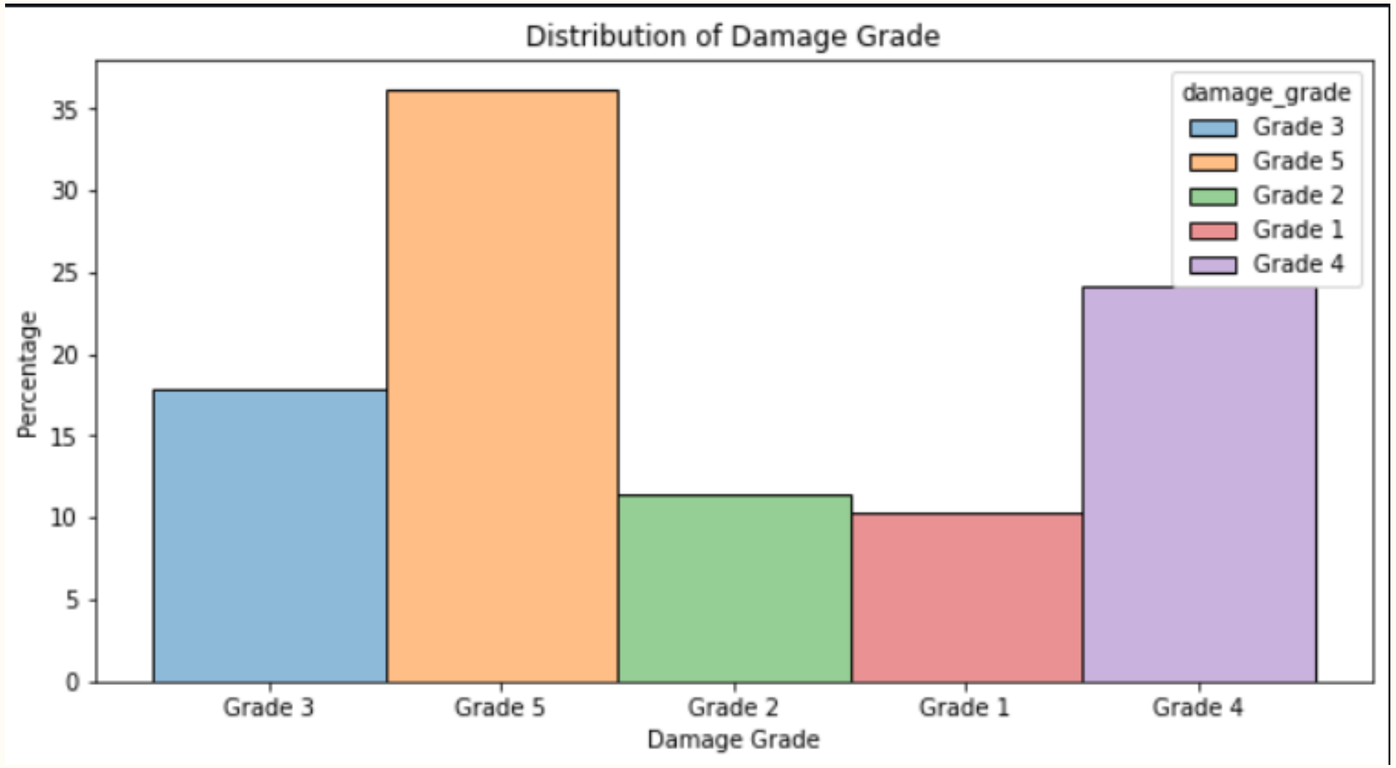
- From Kaggle: **DATASET**
- Data Overview: 762106 rows and 31 columns
- Target: damage_grade(grade 1, 2, 3, 4, 5)

```
df.columns
```

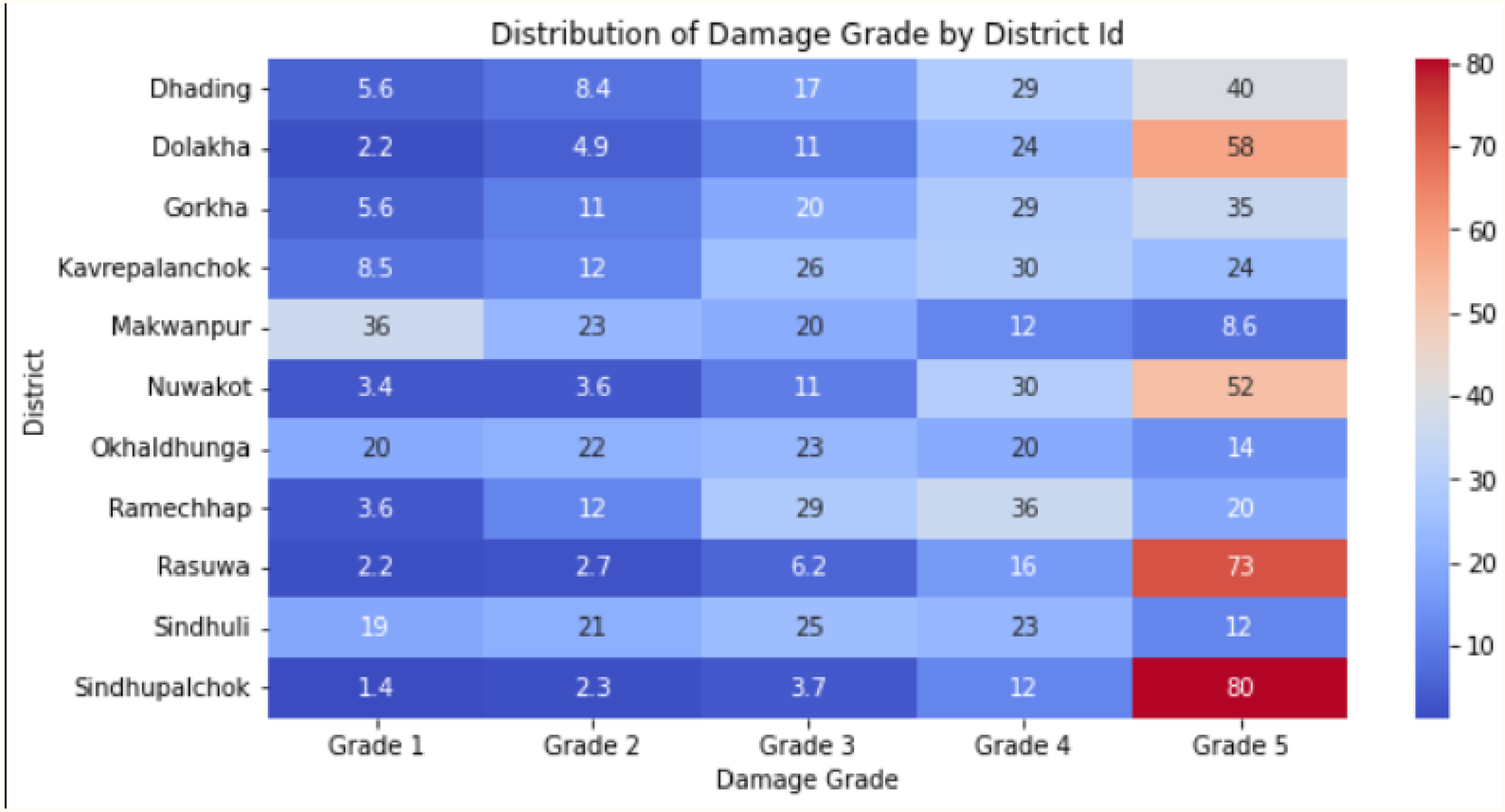
✓ 0.8s

```
Index(['building_id', 'district_id', 'vdcmun_id', 'ward_id',
       'count_floors_pre_eq', 'count_floors_post_eq', 'age_building',
       'plinth_area_sq_ft', 'height_ft_pre_eq', 'height_ft_post_eq',
       'land_surface_condition', 'foundation_type', 'roof_type',
       'ground_floor_type', 'other_floor_type', 'position',
       'plan_configuration', 'has_superstructure_adobe_mud',
       'has_superstructure_mud_mortar_stone', 'has_superstructure_stone_flag',
       'has_superstructure_cement_mortar_stone',
       'has_superstructure_mud_mortar_brick',
       'has_superstructure_cement_mortar_brick', 'has_superstructure_timber',
       'has_superstructure_bamboo', 'has_superstructure_rc_non_engineered',
       'has_superstructure_rc_engineered', 'has_superstructure_other',
       'condition_post_eq', 'damage_grade', 'technical_solution_proposed'],
      dtype='object')
```

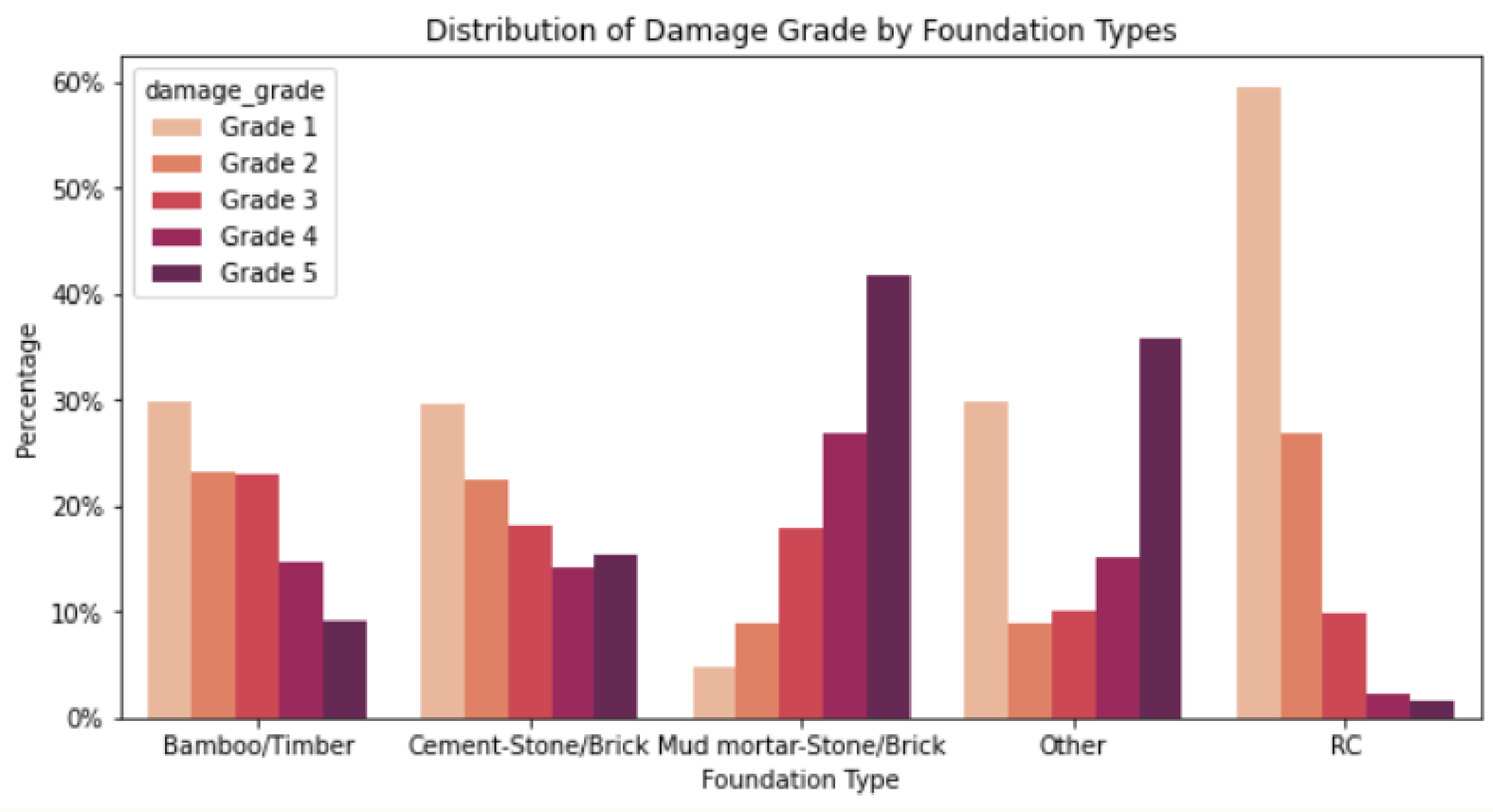
# EXPLORATORY DATA ANALYSIS



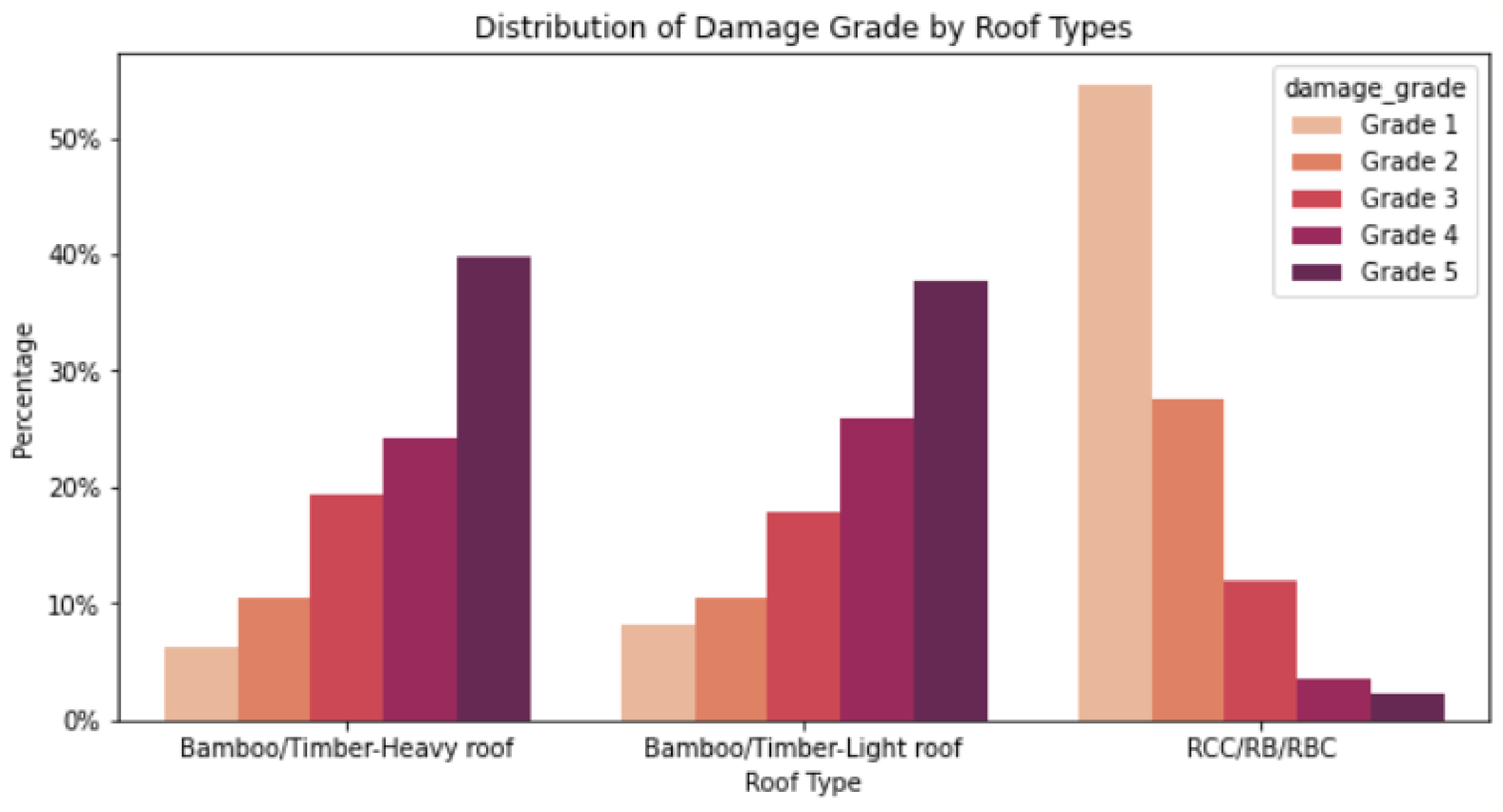

- **The classes seem to be unevenly distributed with most data being of class 'Grade 5'**



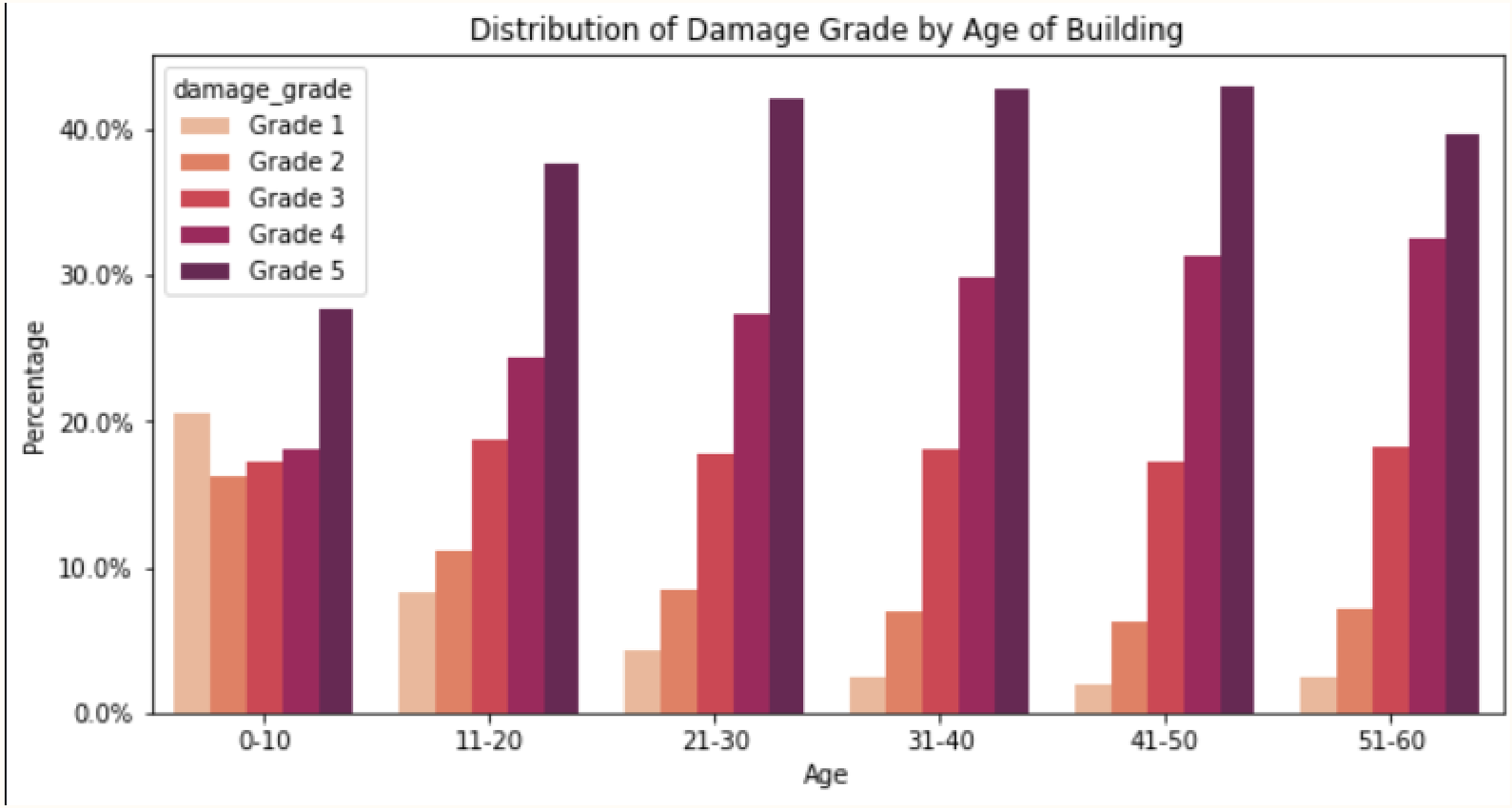

- **A total of 80% of buildings in the Sindhupalchok district were classified as 'Grade 5' damaged buildings.**
- **The least affected among these 11 district seem to be Makwanpur with 69% of building suffering damages below Grade 4**



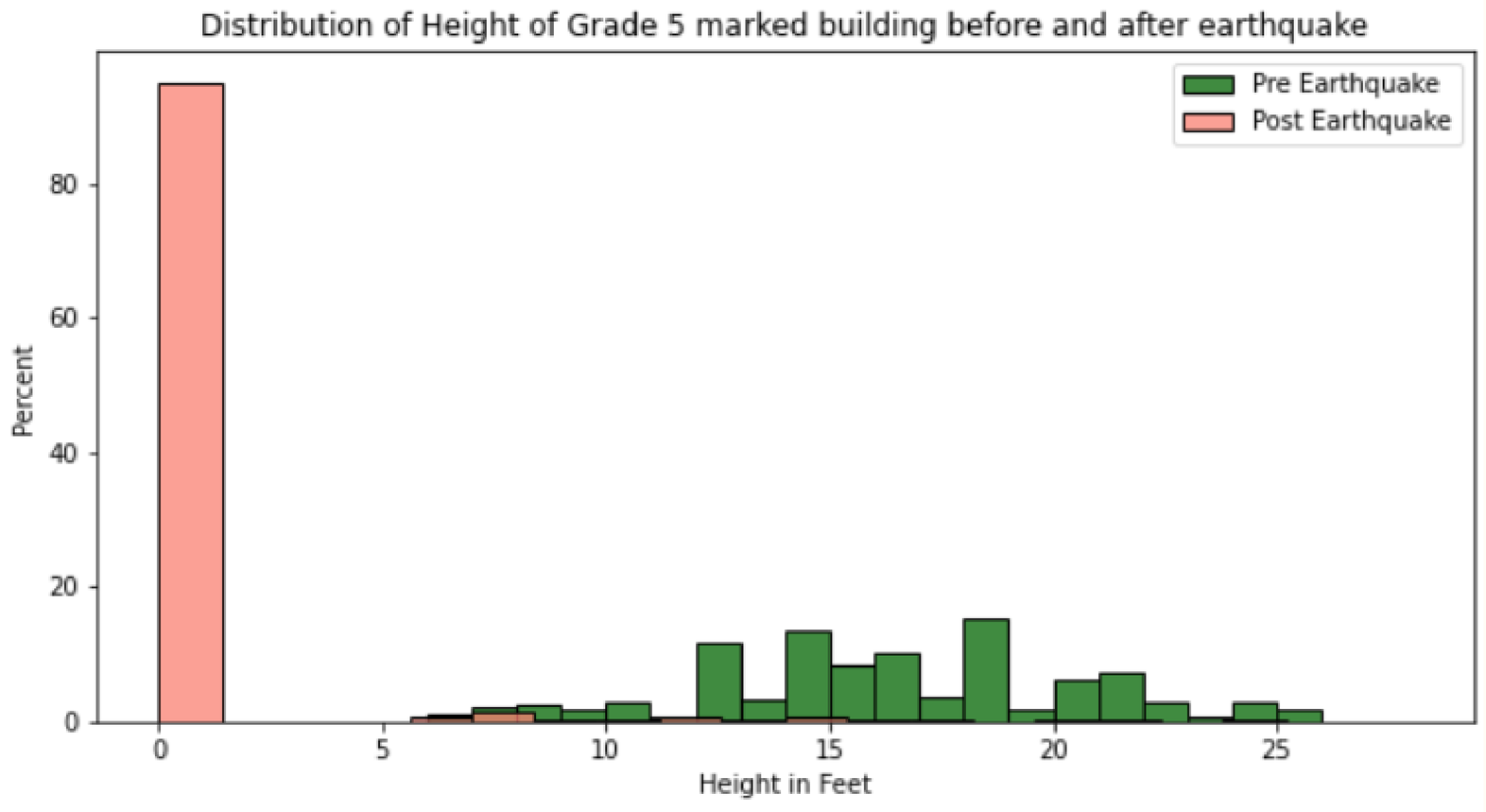

- **Buildings with foundations of Mud-mortar-Stone/Brick suffered the most damage from the earthquake. Almost 80% of these buildings suffered grade 4 or 5 damage.**
- **Buildings with the foundation of RC suffered the least damage from the earthquake.**



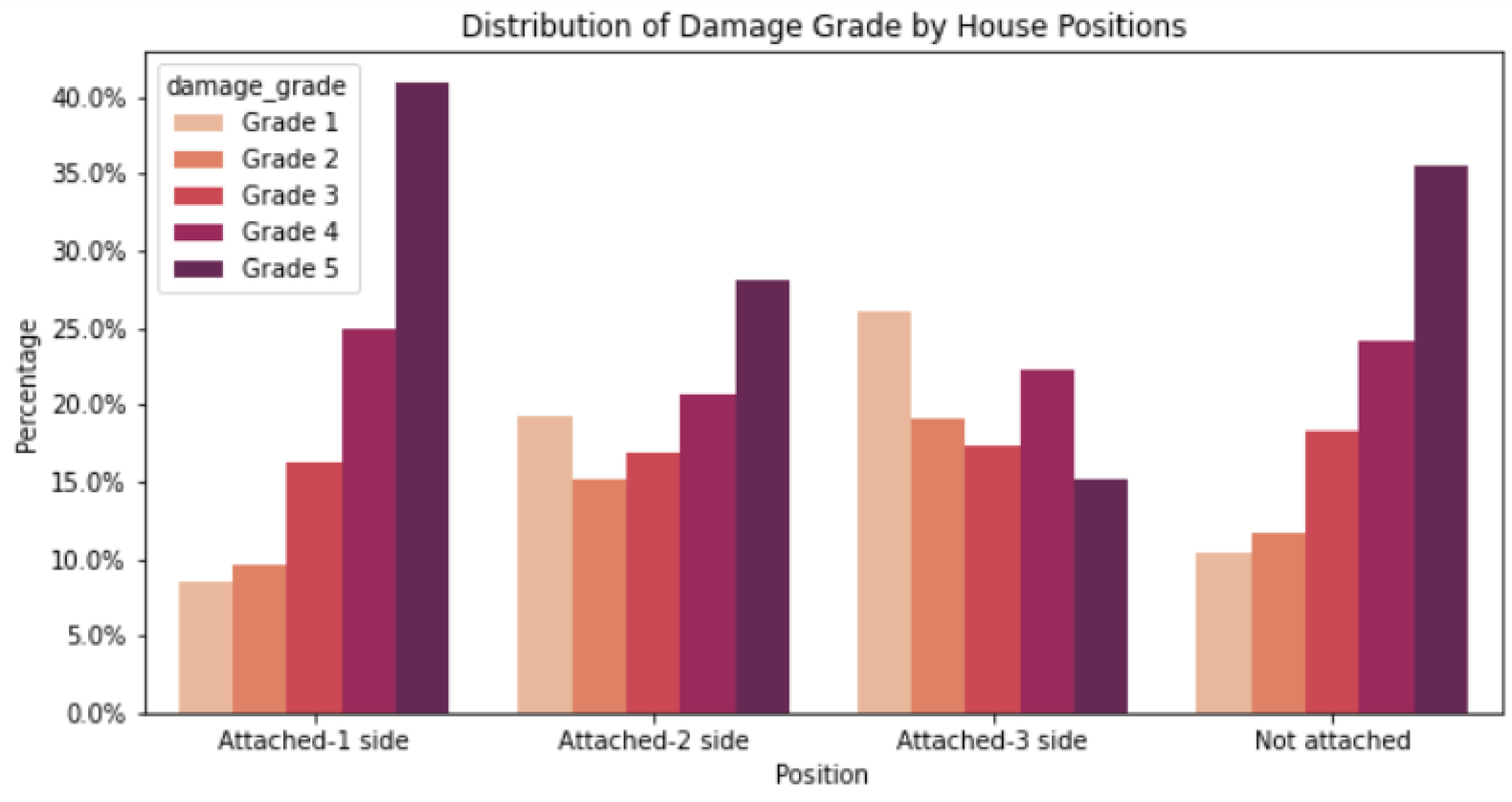

- **Buildings having Bamboo/Timber roofs are prone to serious damage by earthquakes whereas buildings made up of RCC/RB/RBC roof type seem to be resilient to high damage.**



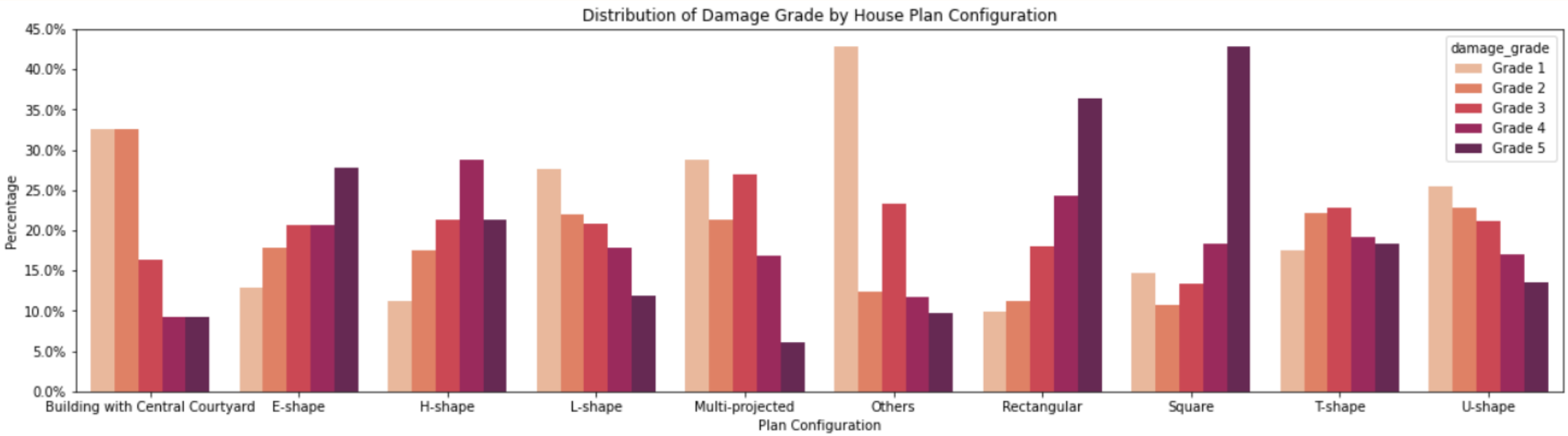

- **Building above the age of 10 seem be highly prone to heavy damages by the earthquake**



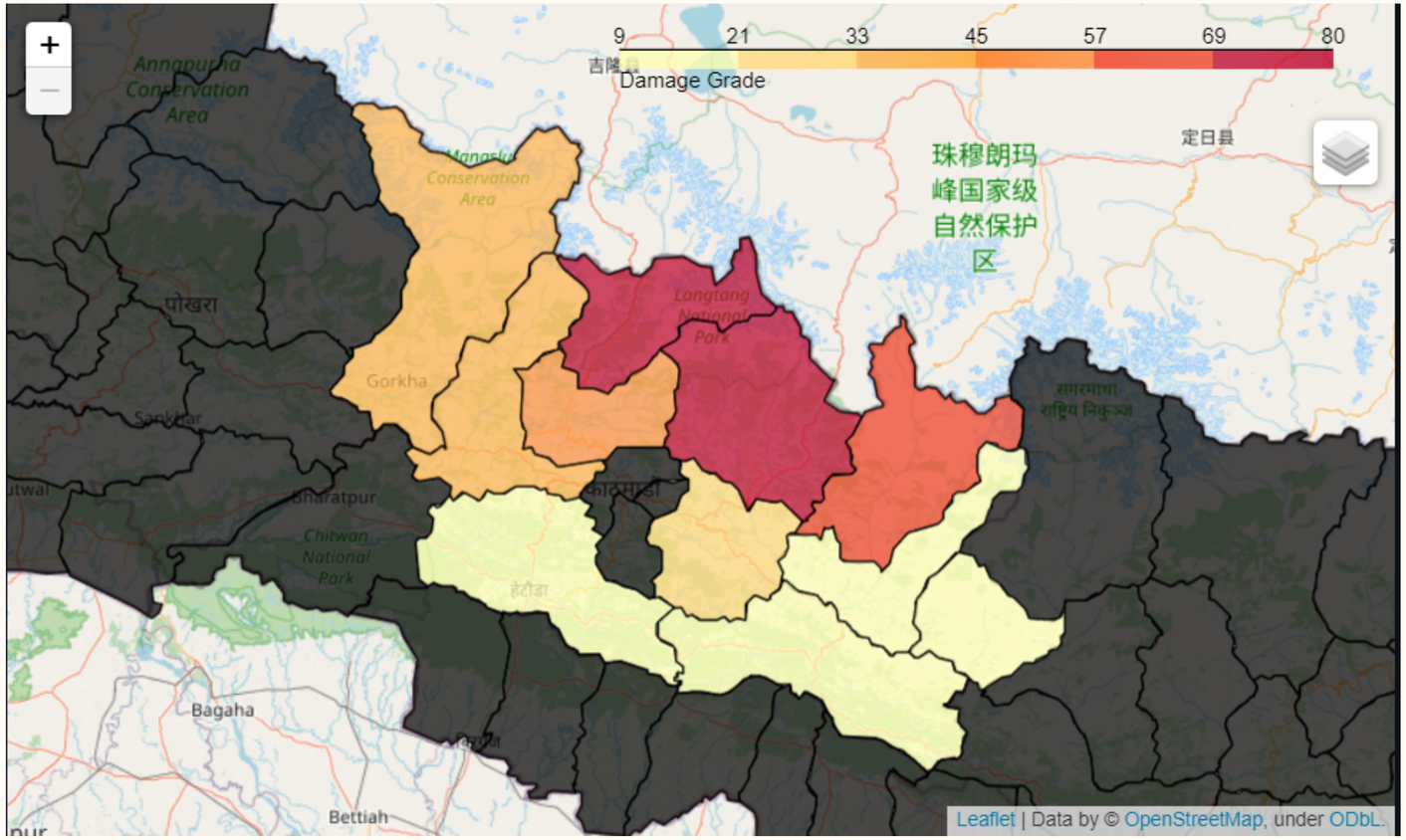

- **Most buildings that were marked Grade 5 damage were reduced to ground by the earthquake**



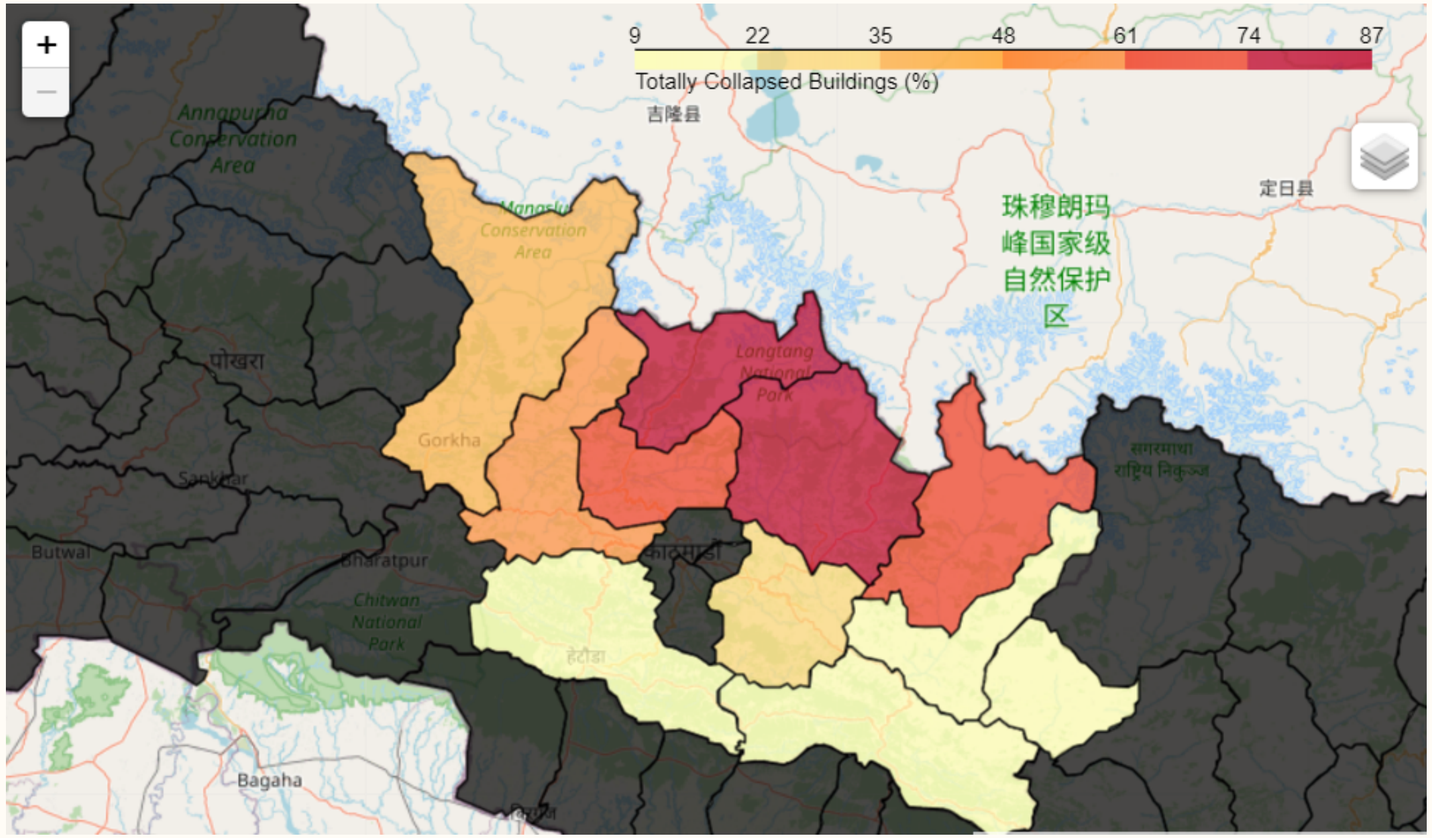

- **Buildings having buildings attached to 3 sides seem to be less affected by the earthquake.**
- **Buildings having building attached to only 1 side seems to be the most affected by the earthquake.**




- **Building with Plan Configuration shaped Rectangular and Square seems to be most prone to high damage from the earthquake**
- **Central-Courtyarded Buildings appear to be less affected by the earthquake**




- **Rasuwa and Sindhupalchowk have most percentage of Grade 5 damaged buildings followed by Dolakha.**




- **Rasuwa and Sindhupalchowk have the most percentage of totally destroyed buildings followed by Dolakha and Nuwakot.**

# DATA PREPROCESSING

# NULL VALUES

```
building_id                              0
district_id                              0
vdcmun_id                                0
ward_id                                  0
count_floors_pre_eq                      0
count_floors_post_eq                     0
age_building                             0
plinth_area_sq_ft                        0
height_ft_pre_eq                         0
height_ft_post_eq                        0
land_surface_condition                   0
foundation_type                          0
roof_type                                0
ground_floor_type                        0
other_floor_type                         0
position                                 1
plan_configuration                       1
has_superstructure_adobe_mud             0
has_superstructure_mud_mortar_stone      0
has_superstructure_stone_flag            0
has_superstructure_cement_mortar_stone   0
has_superstructure_mud_mortar_brick      0
has_superstructure_cement_mortar_brick   0
has_superstructure_timber                0
has_superstructure_bamboo                0
has_superstructure_rc_non_engineered     0
has_superstructure_rc_engineered         0
has_superstructure_other                 0
condition_post_eq                        0
damage_grade                            12
technical_solution_proposed             12
dtype: int64
```

# DATA CLEANING

All rows with null values in target column were dropped as the null count was only 12

```
def cleanDataByRemoving(df):
    df.dropna(inplace=True)
    return df
```

# FEATURE SELECTION

- Damage grade is target variable and rest of columns in data frame as features
- Train test split performed with split ratio of 75% train, 25% test.
- Stratify = y so that both train and test set contains approximately the same percentage of samples of each target class.

```
def selectFeatures(df):
    # Variables for selecting features
    random_seed = 42
    test_ratio = 0.25
    # Damage grade is the required target variable
    y = df["damage_grade"]
    # Technical solution proposed is conflicting with output so trying by dropping it
    # Currently considering all other columns as features
    X = df.drop("damage_grade",axis=1)
    # stratify help to make equal distribution of classes in both train and test
    X_train, X_test, y_train, y_test =  train_test_split(X,y,test_size = test_ratio, random_state = random_seed, stratify = y)
    return X_train, X_test, y_train, y_test, X, y
```

# VARIABLE ENCODING

- Label Encoder for Ordinal variables.
- One hot encoding for Nominal variables

```
def encodeVariables(df):
    # numeric columns
    numeric_cols = [col for col in df.columns if df[col].dtype == 'int64']
    # Categorical ordinal columns
    cat_ordinal=["land_surface_condition","position","damage_grade",
    "technical_solution_proposed"]
    # Nominal ordinal columns
    cat_nominal=["foundation_type","ground_floor_type","other_floor_type",
    "plan_configuration","condition_post_eq","roof_type"]
    df[cat_ordinal] = df[cat_ordinal].apply(LabelEncoder().fit_transform)
    df = pd.get_dummies(df,columns=cat_nominal,prefix=cat_nominal)
    return df
```

# FEATURE ENGINEERING

- Net Floor and Net Height calculated from pre- and post-earthquake data
- Calculated decrement instead of flat value
- High correlation
- Converted to single feature using mean

```
df['net_floor'] = df['count_floors_pre_eq'] - df['count_floors_post_eq']
df['net_height'] = df['height_ft_pre_eq'] - df['height_ft_post_eq']
```

```
# Adding two new columns net_floors and net_height from count_floors and height
df['net_floor'] = df['net_floor']/df.count_floors_pre_eq
df['net_height'] = df['net_height']/df.height_ft_pre_eq
```

# FEATURE REMOVAL

- Unwanted features were removed.
- Building id, District id, vdcmun id, ward id removed
- Count floors and height_ft removed
- net height and net floor removed
- technical_solution_proposed removed.

```
# Removing unwanted columns
df = df.drop(['technical_solution_proposed',
'building_id','district_id','vdcmun_id','ward_id',
'count_floors_post_eq','count_floors_pre_eq',
'height_ft_post_eq','height_ft_pre_eq',
'net_height','net_floor'],axis=1)
```

# FEATURE SCALING

- Standard Scaler used to scale the features i-e X_train and X_test
- This reduced computational complexity and helped to reduce model training time.

```
def scaleFeatures(X_train,X_test):
    scaler = StandardScaler()
    X_train1 = scaler.fit_transform(X_train)
    X_test1 = scaler.transform(X_test)
    return X_train1, X_test1
```

# MODELS

LOGISTIC REGRESSION

ADA BOOST

DECISION TREE

GRADIENT BOOST

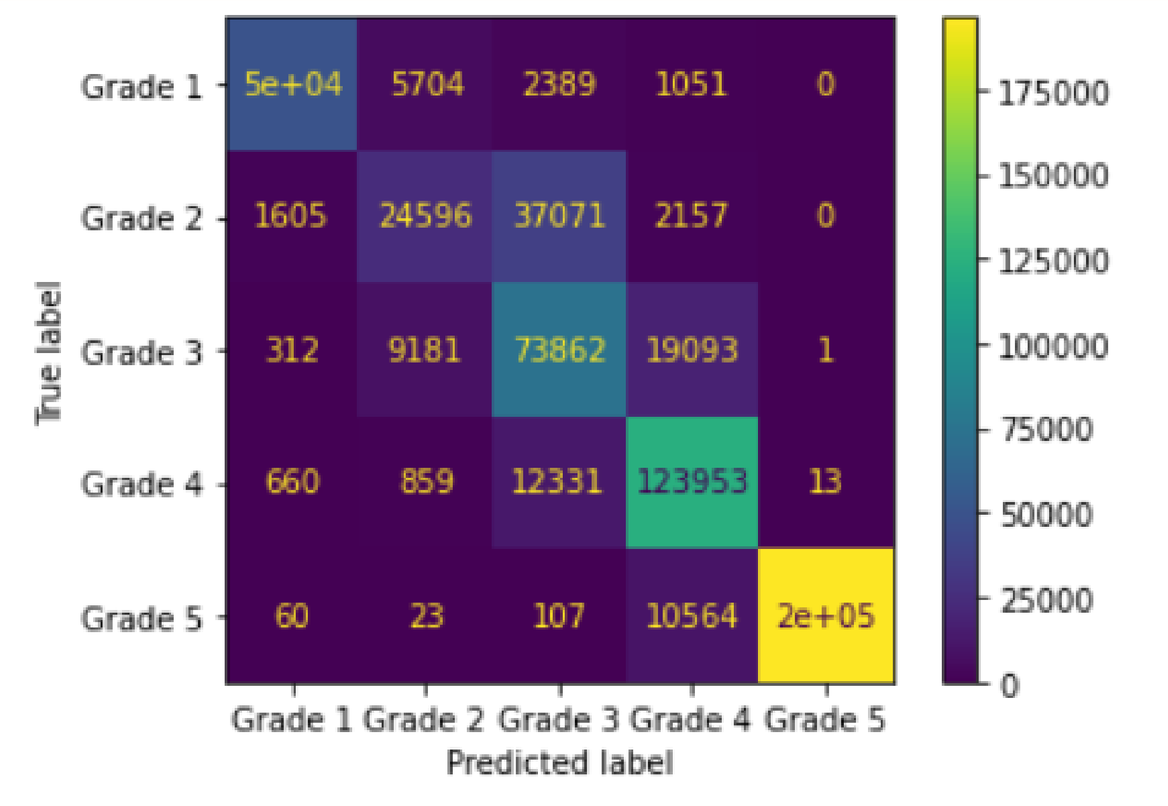
RANDOM FOREST

XG BOOST

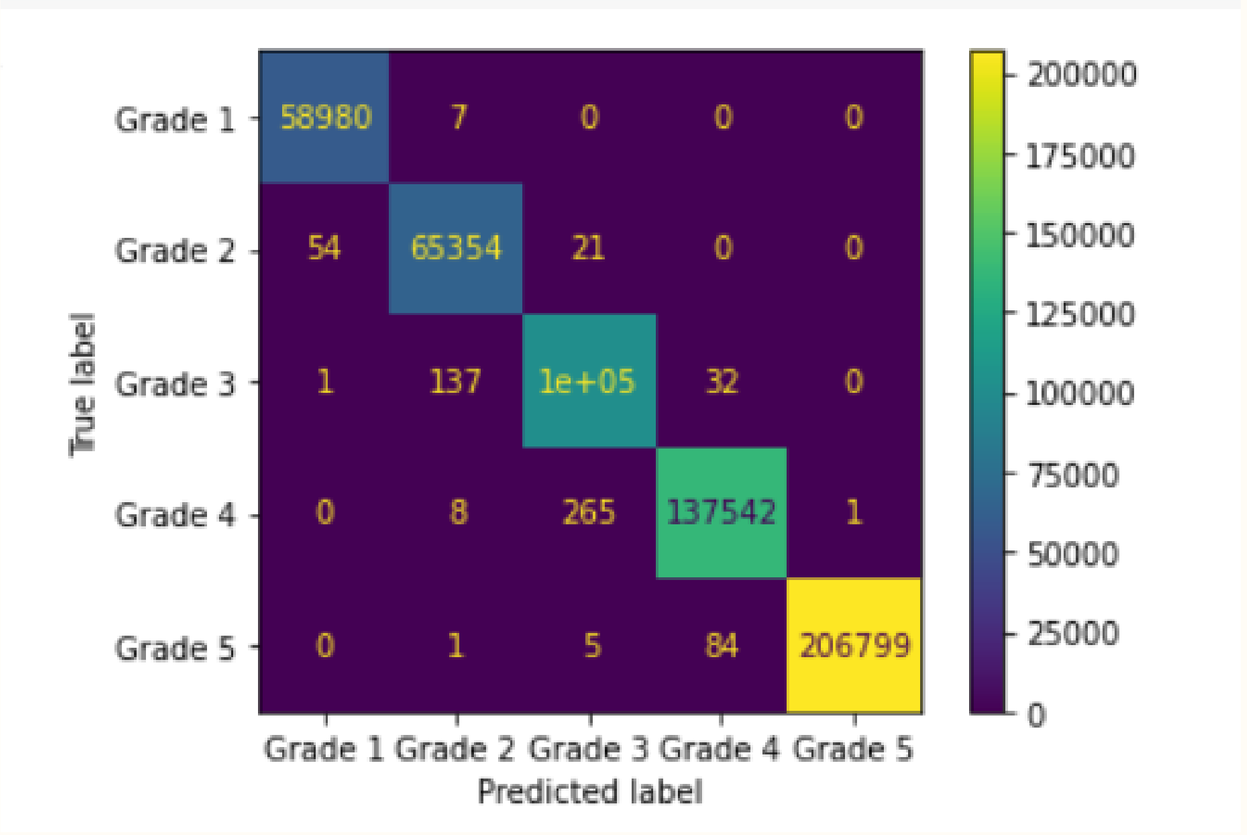
# PARAMETERS

| | Model Name | Accuracy | Precision | Recall | Test F1 Score | Train F1 Score |
|---|---|---|---|---|---|---|
| 0 | Logistic Regression | 0.819304 | 0.825867 | 0.819304 | 0.816788 | 0.816722 |
| 1 | Decision Tree | 0.873040 | 0.872883 | 0.873040 | 0.872958 | 0.998922 |
| 2 | Random Forest | 0.903503 | 0.905435 | 0.903503 | 0.904199 | 0.998912 |
| 3 | Gradient Boosting | 0.883453 | 0.889472 | 0.883453 | 0.884711 | 0.884191 |
| 4 | Ada Boost | 0.796677 | 0.823926 | 0.796677 | 0.778001 | 0.777413 |
| 5 | XG Boost | 0.881532 | 0.888086 | 0.881532 | 0.882907 | 0.882235 |

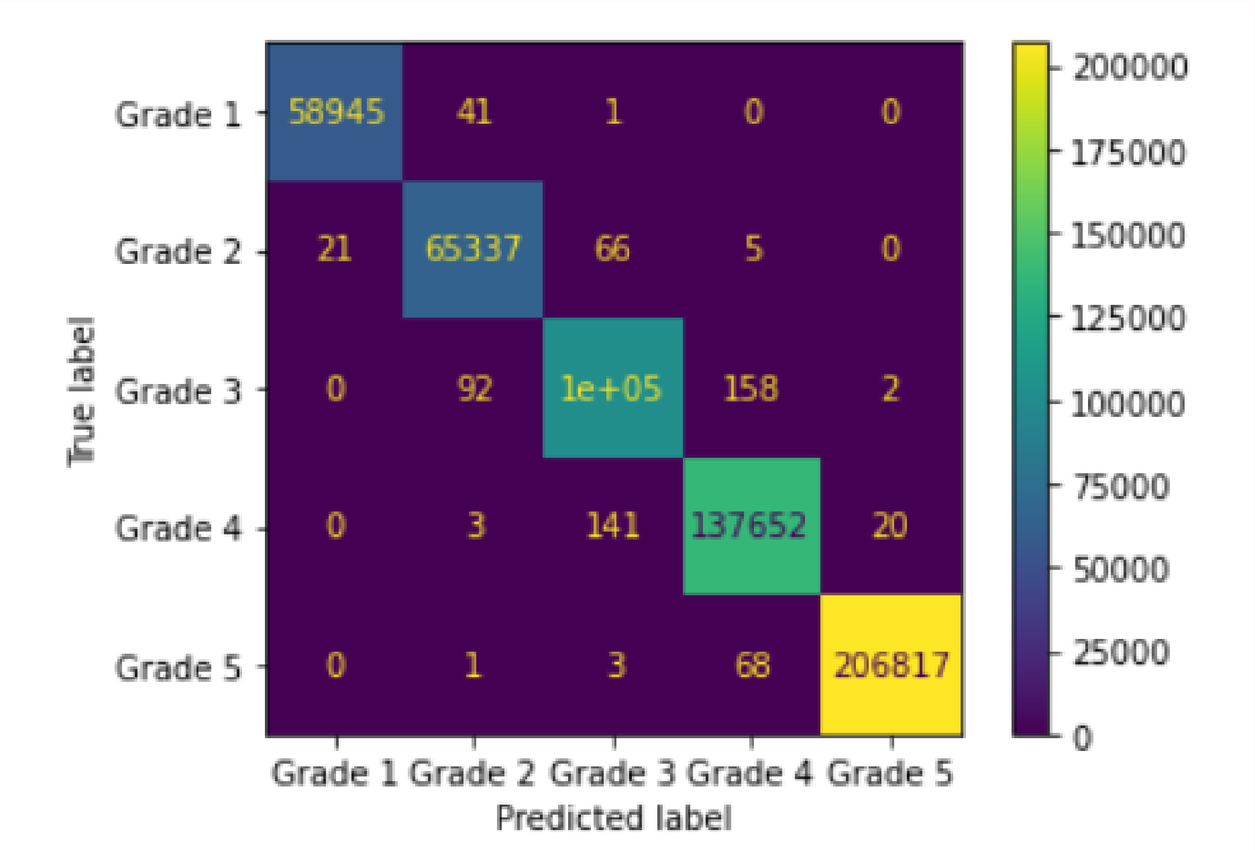
# LOGISTIC REGRESSION



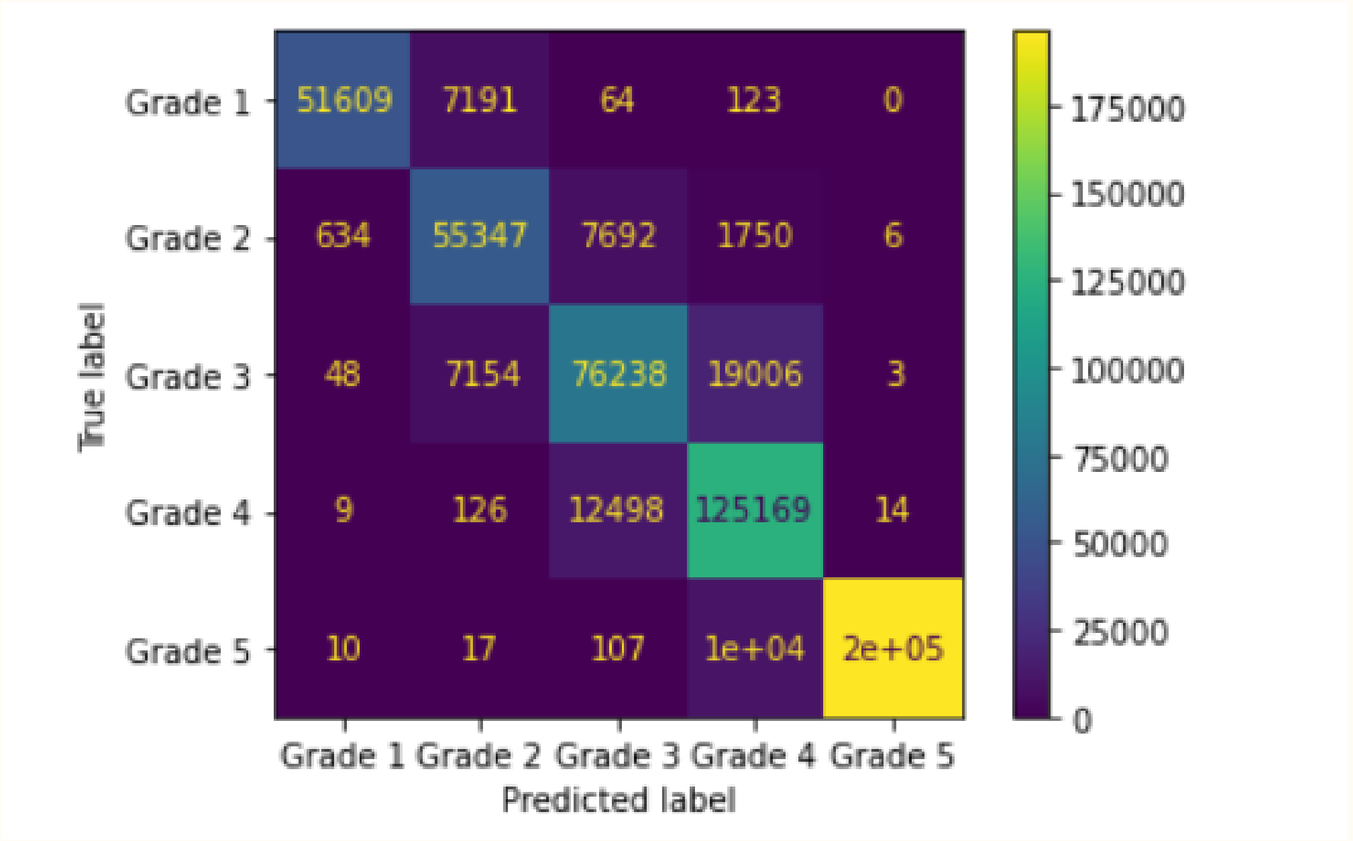
# DECISION TREE



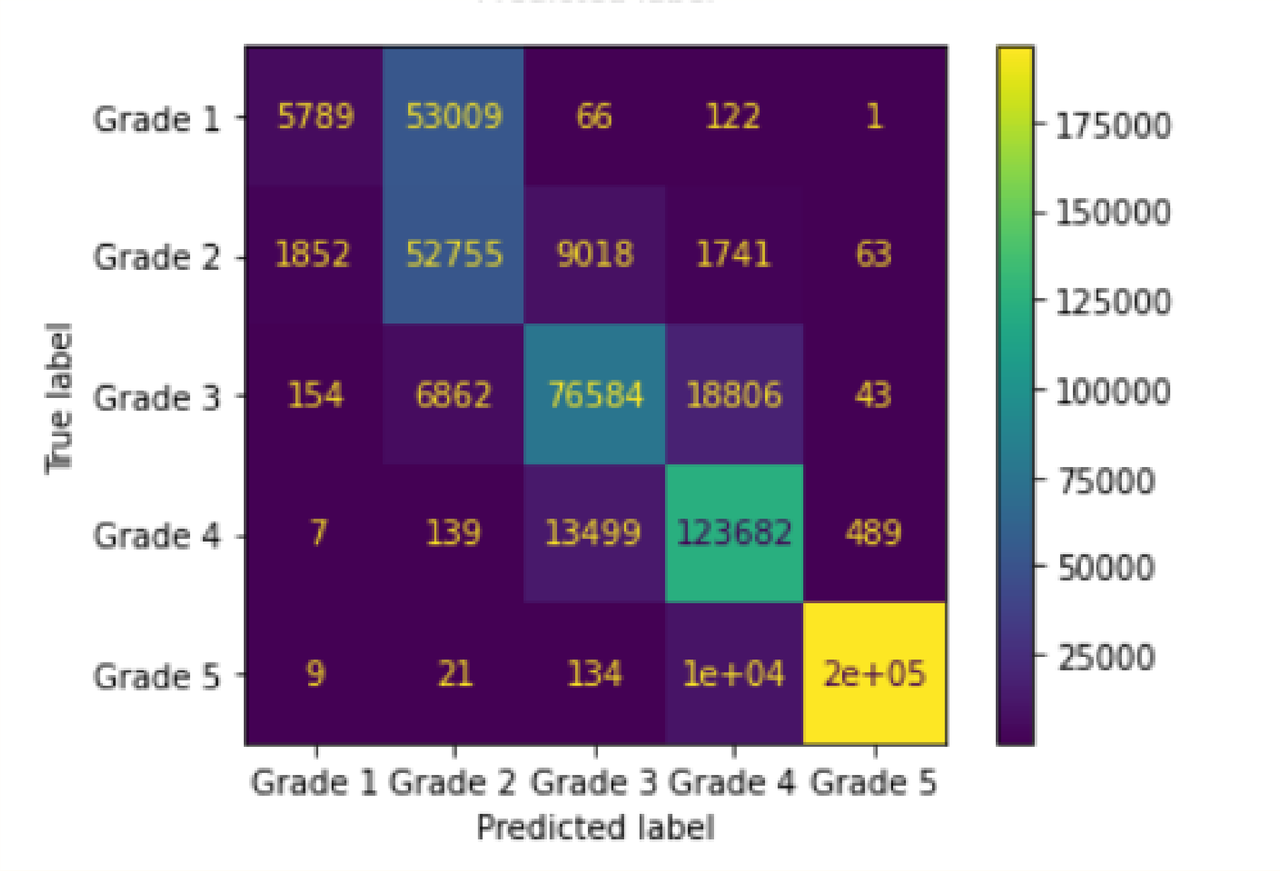
# RANDOM FOREST



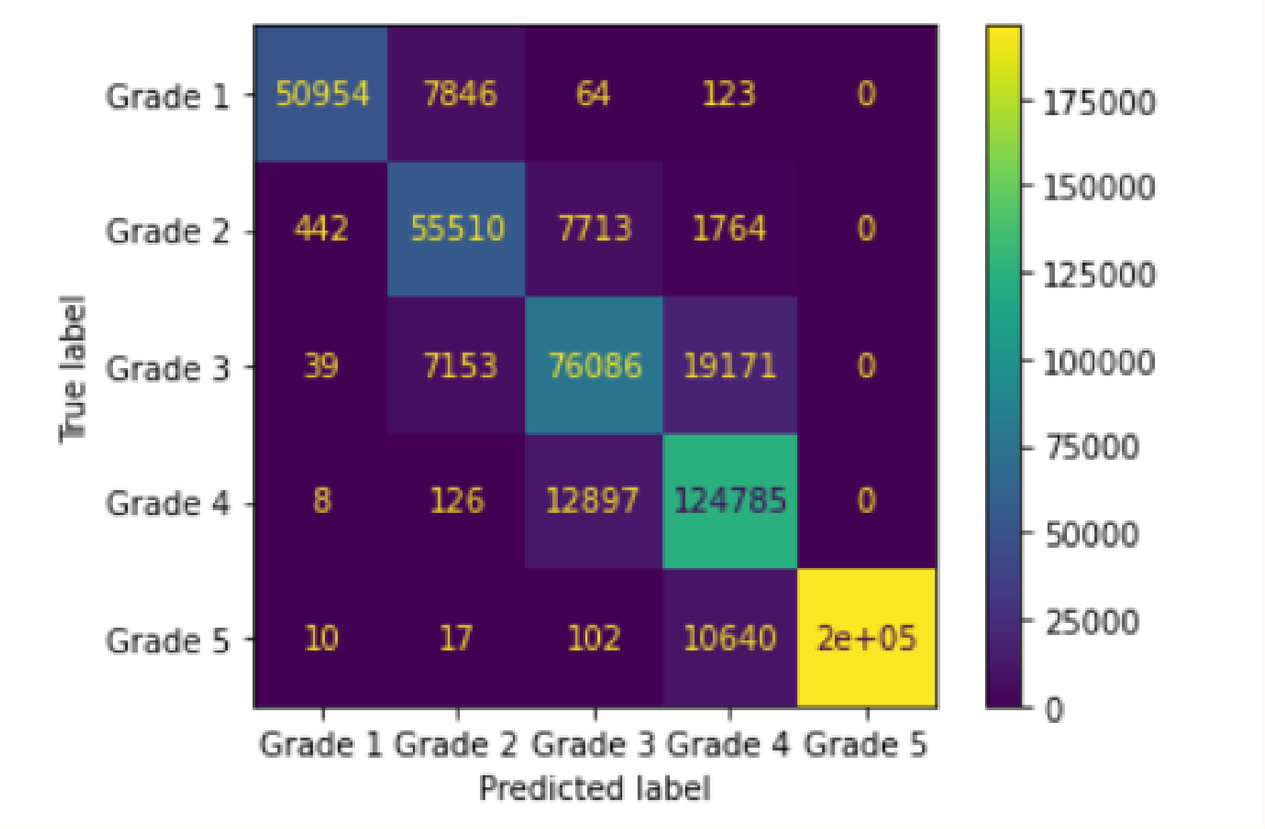
# GRADIENT BOOST



# ADA BOOST


Grade 1
Grade 2
Grade 3
Grade 4
Grade 5
True label
5789
53009
66
122
1
1852
52755
9018
1741
63
154
6862
76584
18806
43
7
139
13499
123682
489
9
21
134
1e+04
2e+05
175000
150000
125000
100000
75000
50000
25000
Grade 1 Grade 2 Grade 3 Grade 4 Grade 5
Predicted label

# XG BOOST


Grade 1
Grade 2
Grade 3
Grade 4
Grade 5
True label
50954
7846
64
123
0
442
55510
7713
1764
0
39
7153
76086
19171
0
8
126
12897
124785
0
10
17
102
10640
2e+05
Grade 1 Grade 2 Grade 3 Grade 4 Grade 5
Predicted label
175000
150000
125000
100000
75000
50000
25000
0

# PARAMETERS

## Considering Top 10 features(columns) only

```
Index(['height_ft_post_eq', 'technical_solution_proposed',
       'count_floors_post_eq', 'condition_post_eq_Not damaged',
       'has_superstructure_mud_mortar_stone',
       'condition_post_eq_Damaged-Rubble unclear',
       'foundation_type_Mud mortar-Stone/Brick', 'ground_floor_type_RC',
       'condition_post_eq_Damaged-Rubble clear',
       'has_superstructure_cement_mortar_brick'],
      dtype='object')
```

# Considering Top 10 features(columns) only

| | Model Name | Accuracy | Precision | Recall | Test F1 Score | Train F1 Score |
|---|---|---|---|---|---|---|
| 0 | Decision Tree | 0.880503 | 0.886974 | 0.880503 | 0.881876 | 0.882462 |
| 1 | Random Forest | 0.880750 | 0.887232 | 0.880750 | 0.882120 | 0.882464 |
| 2 | Gradient Boosting | 0.880955 | 0.887499 | 0.880955 | 0.882357 | 0.881785 |
| 3 | XG Boost | 0.880955 | 0.887499 | 0.880955 | 0.882357 | 0.881782 |

# HYPERPARAMETER TUNING

## Considering Random Forest only

Finally best hyperparameter obtained for Random Forest model was:

n_estimators = 300

min_samples_split = 10

min_samples_leaf = 1

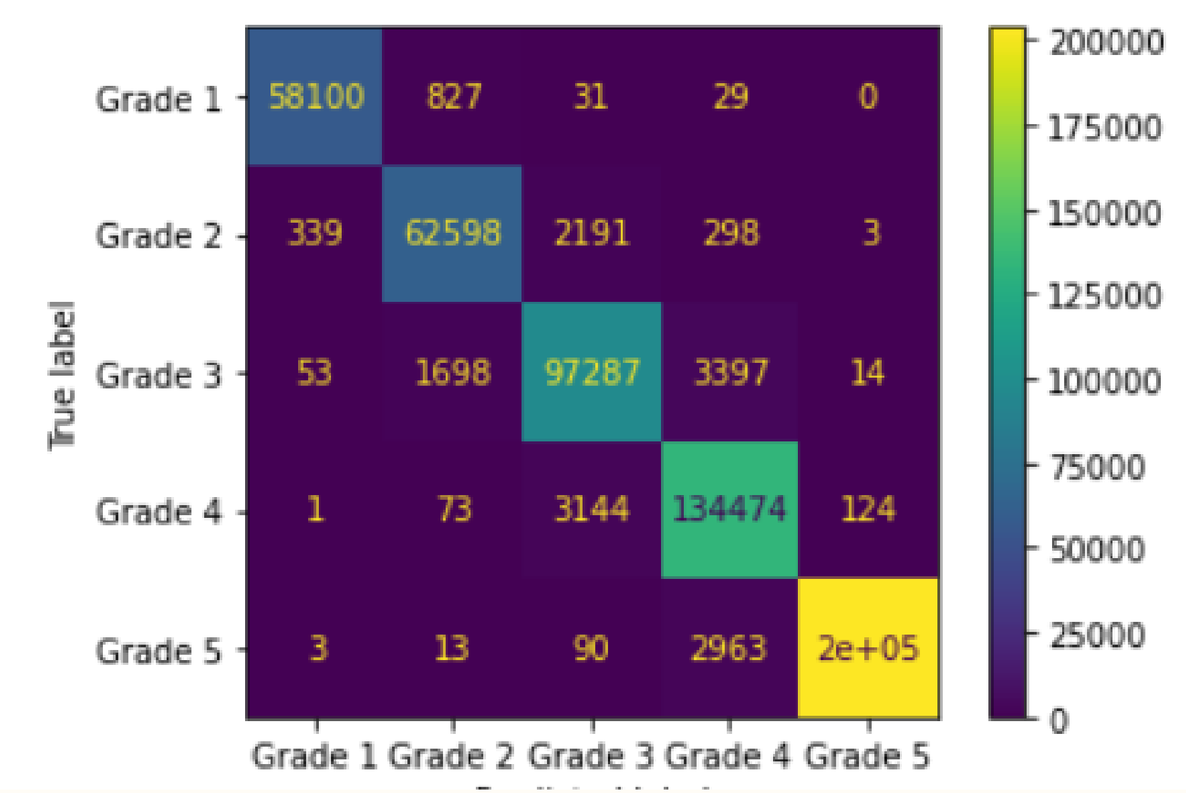
max_features = auto

max_depth = 100

bootstrap = False

| | Model Name | Accuracy | Precision | Recall | Test F1 Score | Train F1 Score |
|---|---|---|---|---|---|---|
| **0** | Random Forest | 0.906426 | 0.908614 | 0.906426 | 0.907169 | 0.973333 |

# Considering Random Forest only

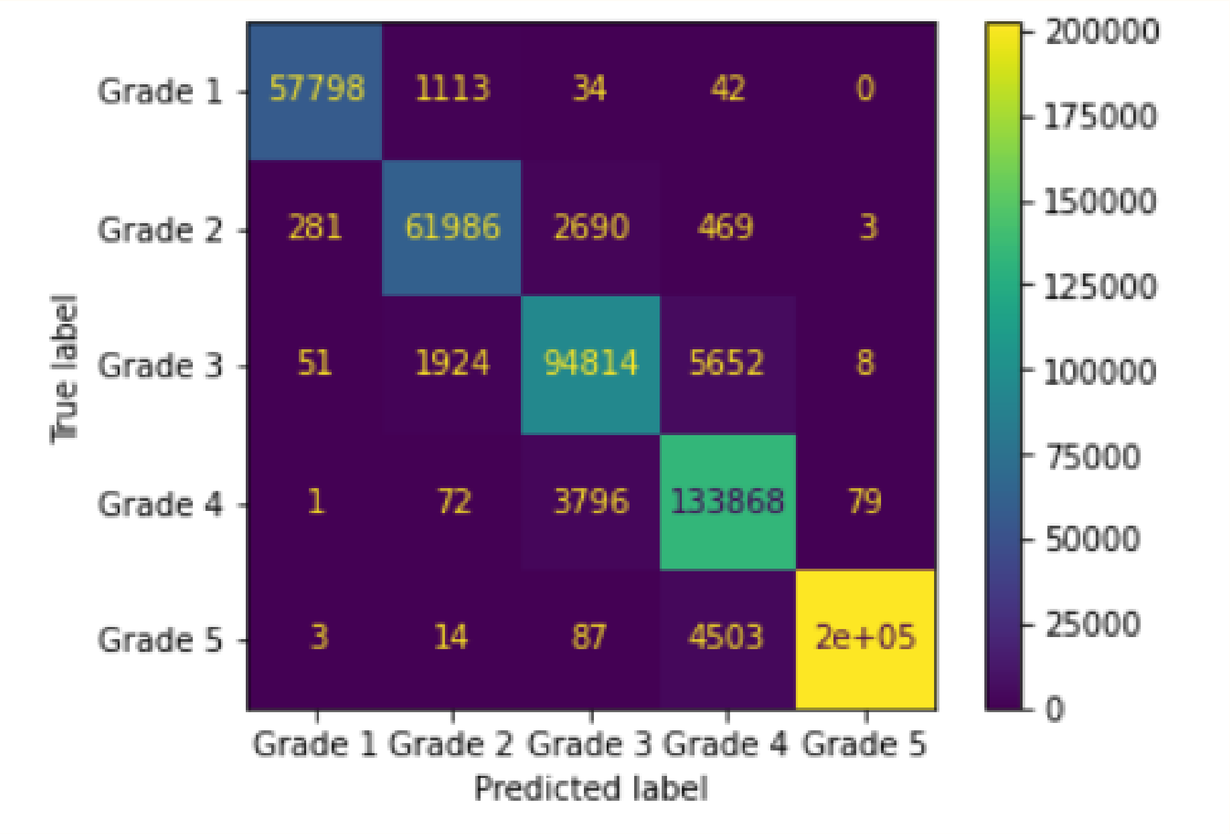


## Considering Random Forest only

{'bootstrap': False,
'max_depth': 50,
'min_samples_leaf': 2,
'min_samples_split': 2,
'n_estimators': 200}

| | Model Name | Accuracy | Precision | Recall | Test F1 Score | Train F1 Score |
|---|---|---|---|---|---|---|
| **0** | Random Forest | 0.905445 | 0.907978 | 0.905445 | 0.906206 | 0.963743 |

# Considering Random Forest only



# MODEL OPTIMIZATION

- Under-sampling and Oversampling
- Hyperparameter Tuning
- PCA
- Deployment

*All methods were performed on Random Forest Classifier*

# UNDERSAMPLING

- 2 methods used for undersampling
  - Random undersampling
  - Nearmiss undersampling.
- Data undersampled so all the classes have equal 58635 rows

```
def underSampleData(X_train,y_train,X_test,y_test):
    # random under sampling
    rus = RandomUnderSampler(random_state=42)
    X_train_rus, y_train_rus = rus.fit_resample(X_train,y_train)
    # Check the number of records after over sampling
    print(sorted(Counter(y_train_rus).items()))
    # Nearmiss under sampling
    nearmiss = NearMiss(version=3)
    X_train_nearmiss, y_train_nearmiss = nearmiss.fit_resample(X_train, y_train)
    # Check the number of records after over sampling
    print(sorted(Counter(y_train_nearmiss).items()))
    return X_train_rus, y_train_rus, X_train_nearmiss, y_train_nearmiss
```

# OVERSAMPLING

- 2 methods used for oversampling
  - Random oversampling
  - SMOTE undersampling.
- Data oversampled so all the classes have equal 206784 rows

```
def overSampleData(X_train,y_train):
    # random over sampling
    ros = RandomOverSampler(random_state=42)
    X_train_ros, y_train_ros = ros.fit_resample(X_train,y_train)
    # Check the number of records after over sampling
    print(sorted(Counter(y_train_ros).items()))
    smote = SMOTE(random_state=42)
    X_train_smote, y_train_smote= smote.fit_resample(X_train, y_train)
    # Check the number of records after over sampling
    print(sorted(Counter(y_train_smote).items()))
    return X_train_ros, y_train_ros, X_train_smote,y_train_smote
```

# SAMPLING RESULTS

| Model Name | Accuracy_Test | Accuracy_Train | Precision_Test | Precision_Train | Recall_Test | Recall_Train | F1_Score_Test | F1_Score_Train |
|---|---|---|---|---|---|---|---|---|
| Normal | 0.713532 | 0.714039 | 0.717065 | 0.718432 | 0.713532 | 0.714039 | 0.698504 | 0.699440 |
| Random Oversampling | 0.713495 | 0.662418 | 0.719238 | 0.666584 | 0.713495 | 0.662418 | 0.708035 | 0.655470 |
| SMOTE Oversampling | 0.716540 | 0.661617 | 0.722646 | 0.663711 | 0.716540 | 0.661617 | 0.714315 | 0.658882 |
| Random Undersampling | 0.713894 | 0.662895 | 0.720699 | 0.668616 | 0.713894 | 0.662895 | 0.708431 | 0.655997 |
| Near Miss Undersampling | 0.693913 | 0.741043 | 0.698761 | 0.714536 | 0.693913 | 0.741043 | 0.670236 | 0.707567 |

- Random under and over sampling took few seconds to complete
- SMOTE oversampling took about 13 minutes and nearmiss undersampling took about 12 minutes.

# RESULTS OF FEATURE ENGINEERING AND REMOVAL

| Model Name | Accuracy_Test | Accuracy_Train | Precision_Test | Precision_Train | Recall_Test | Recall_Train | F1_Score_Test | F1_Score_Train |
|---|---|---|---|---|---|---|---|---|
| Normal | 0.731146 | 0.749855 | 0.737879 | 0.757903 | 0.731146 | 0.749855 | 0.729433 | 0.748666 |
| Random Oversampling | 0.722761 | 0.735611 | 0.733074 | 0.746924 | 0.722761 | 0.735611 | 0.722239 | 0.734456 |
| Random Undersampling | 0.721365 | 0.703191 | 0.733419 | 0.715804 | 0.721365 | 0.703191 | 0.720206 | 0.700921 |

- It was observed removing the technical solution proposed feature reduced the overall score of the model from 90% to 72% as it has highest correlation to the target variable.

# HYPERPARAMETER OPTIMIZATION

- Hyperparameter optimization to reduce overfitting.
- 2 methods used:
  - Grid Search and Randomized Search algorithms
  - Manually changing hyperparameters
- Using algorithms took a long time, about 6-12 hrs.
- Better scores obtained from manual hyperparameter tuning.
- Best Parameters Found:
  - n_estimators=150,min_samples_split=5,min_samples_leaf=5,max_depth=30, random_state=RANDOM_STATE
  - This provided test f1_score of 77.89 and train f1_score of 82.

# HYPERPARAMETER OPTIMIZATION

- Hyperparameter optimization to reduce overfitting.

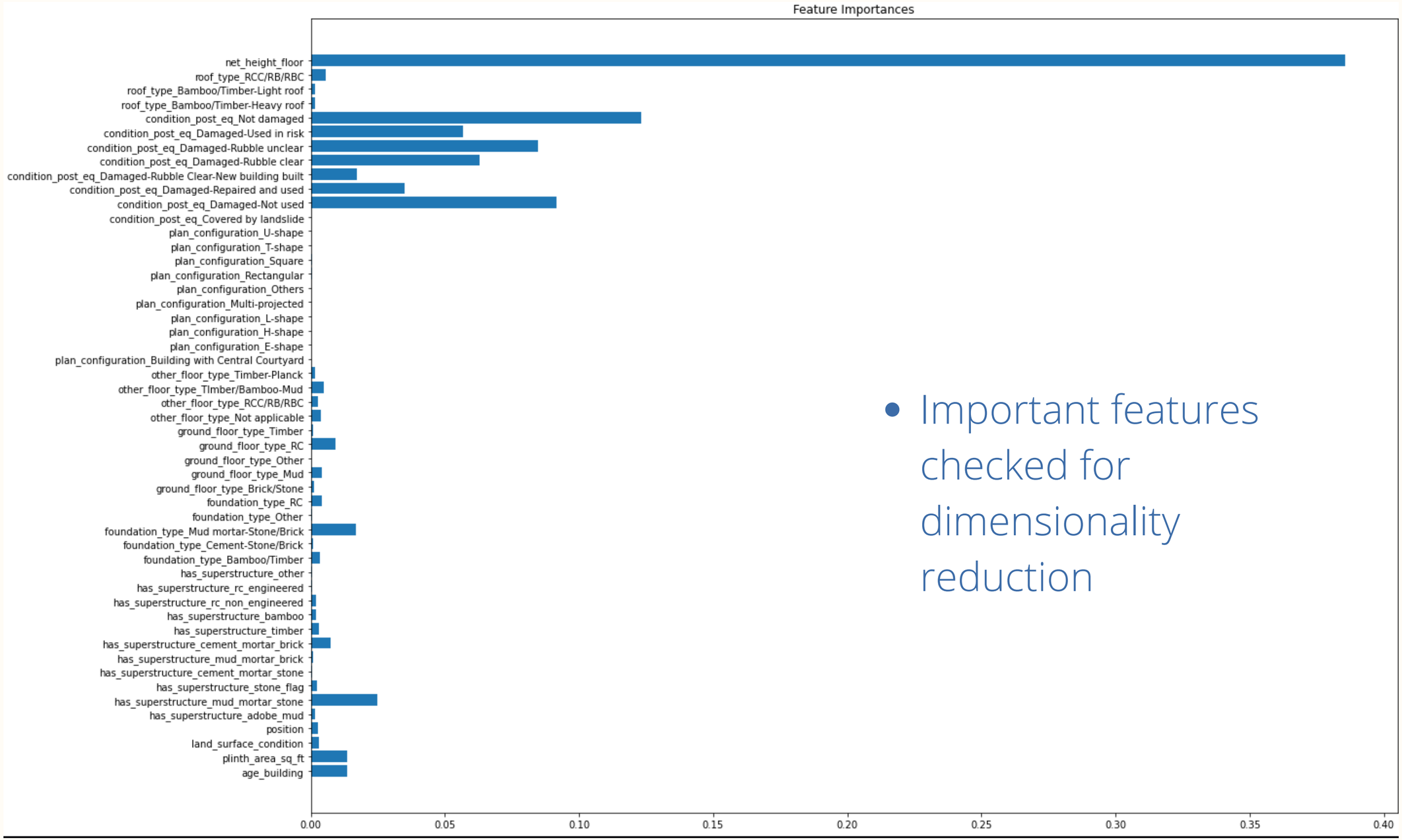
| Model Name | Accuracy_Test | Accuracy_Train | Precision_Test | Precision_Train | Recall_Test | Recall_Train | F1_Score_Test | F1_Score_Train |
|---|---|---|---|---|---|---|---|---|
| Normal | 0.773745 | 0.996336 | 0.775141 | 0.996337 | 0.773745 | 0.996336 | 0.773801 | 0.996336 |
| Random Oversampling | 0.769378 | 0.996662 | 0.772624 | 0.996664 | 0.769378 | 0.996662 | 0.770718 | 0.996662 |
| SMOTE Oversampling | 0.773299 | 0.997572 | 0.776593 | 0.997572 | 0.773299 | 0.997572 | 0.774555 | 0.997572 |
| Random Undersampling | 0.761726 | 0.997138 | 0.771019 | 0.997138 | 0.761726 | 0.997138 | 0.764857 | 0.997138 |
| Near Miss Undersampling | 0.701770 | 0.999008 | 0.700114 | 0.999008 | 0.701770 | 0.999008 | 0.698437 | 0.999008 |

Outputs when no hyperparameters used

| Model Name | Accuracy_Test | Accuracy_Train | Precision_Test | Precision_Train | Recall_Test | Recall_Train | F1_Score_Test | F1_Score_Train |
|---|---|---|---|---|---|---|---|---|
| Normal | 0.774685 | 0.822207 | 0.777916 | 0.827563 | 0.774685 | 0.822207 | 0.772696 | 0.821312 |
| Random Oversampling | 0.774391 | 0.860454 | 0.781040 | 0.863030 | 0.774391 | 0.860454 | 0.776470 | 0.860974 |
| Random Undersampling | 0.764665 | 0.801245 | 0.773913 | 0.806666 | 0.764665 | 0.801245 | 0.767060 | 0.802177 |

Outputs when hyperparameters tuned

# CHECKING IMPORTANT FEATURES




- Important features checked for dimensionality reduction

# PRINCIPAL COMPONENT ANALYSIS(PCA)

- PCA done to reduce the dimensionality
- Slight overfitting seen after performing PCA

```
from sklearn.decomposition import PCA
pca = PCA(n_components=22)
X_train_normal_pca = pca.fit_transform(X_train_normal)
X_test_normal_pca = pca.transform(X_test_normal)
```

| Model Name | Accuracy_Test | Accuracy_Train | Precision_Test | Precision_Train | Recall_Test | Recall_Train | F1_Score_Test | F1_Score_Train |
|---|---|---|---|---|---|---|---|---|
| Normal | 0.725742 | 0.842372 | 0.728641 | 0.850014 | 0.725742 | 0.842372 | 0.721893 | 0.842096 |
| Random Oversampling | 0.716029 | 0.889136 | 0.722765 | 0.891399 | 0.716029 | 0.889136 | 0.716801 | 0.889784 |
| Random Undersampling | 0.712458 | 0.830180 | 0.723755 | 0.835038 | 0.712458 | 0.830180 | 0.714128 | 0.831488 |

# DEPLOYMENT

- Best model saved in pickle format
- A simple web app developed using Django for user input
- Some data preprocessing done to convert the data from web app into form suitable for model prediction.
- Currently, model hosted only in localhost.

# SAVING MODELS

```
def save_model(model,name):
    pickle.dump(model,open(f'{name}.pkl','wb'))
    print(f'{name} model saved successfully')
```

```
save_model(forest1,'normal_model')
```

```
normal_model model saved successfully
```

# WEB APP DEMO


EQDMGPRED
Home
About Project
Earthquake Building Damage Prediction
AI based prediction system
Predict Damage Grade

# CONCLUSION

- Hence, in this project various machine learning concepts were researched and implemented, and a simple web app was developed to predict the earthquake damage to buildings.

- Overall code can be found in GitHub:
- **JuJu2181/earthquake-damage-predictor: A ML model to predict building damages (github.com)**

- Demo video can be found here: **DEMO VIDEO**

- Some problems we faced are uneven distribution of data, irrelevant features, overfitting, computational time.

# THANK YOU