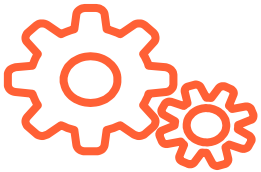
Modelling Earthquake Damage

CZ1015 Introduction to Data Science and Artificial Intelligence

Mini-Project Presentation

Chua Chiah Soon, Garg Astha, Ho Wei Yi Stanley, Nurul Afiqah Binte Arisfadzillah

Problem Statements

Data Exploration and Visualization	Classification model training and testing	Data Exploration and Visualization	Classification model training and testing
			
What are the most important factors which affect the damage grade of the buildings?	Predict the damage grade of the buildings by training the models based on the given dataset.	What are the ideal materials to be used in the construction that would suffer least damage?	Create your own building. Predict the damage it might suffer in case of earthquake.

Description of the Dataset

The dataset was obtained from DRIVENDATA

The dataset consists of 39 columns (excluding building_id).

There is data available for 260,601 buildings located in Nepal.

There are no null or NaN values.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 260601 entries, 0 to 260600
Data columns (total 39 columns):
geo_level_1_id                          260601 non-null int64
geo_level_2_id                          260601 non-null int64
geo_level_3_id                          260601 non-null int64
count_floors_pre_eq                     260601 non-null int64
age                                     260601 non-null int64
area_percentage                         260601 non-null int64
height_percentage                       260601 non-null int64
land_surface_condition                  260601 non-null object
foundation_type                         260601 non-null object
roof_type                               260601 non-null object
ground_floor_type                       260601 non-null object
other_floor_type                        260601 non-null object
position                                260601 non-null object
plan_configuration                      260601 non-null object
has_superstructure_adobe_mud            260601 non-null int64
has_superstructure_mud_mortar_stone     260601 non-null int64
has_superstructure_stone_flag           260601 non-null int64
has_superstructure_cement_mortar_stone  260601 non-null int64
has_superstructure_mud_mortar_brick     260601 non-null int64
has_superstructure_cement_mortar_brick  260601 non-null int64
has_superstructure_timber               260601 non-null int64
has_superstructure_bamboo               260601 non-null int64
has_superstructure_rc_non_engineered    260601 non-null int64
has_superstructure_rc_engineered        260601 non-null int64
has_superstructure_other                260601 non-null int64
legal_ownership_status                  260601 non-null object
count_families                          260601 non-null int64
has_secondary_use                       260601 non-null int64
has_secondary_use_agriculture           260601 non-null int64
has_secondary_use_hotel                 260601 non-null int64
has_secondary_use_rental                260601 non-null int64
has_secondary_use_institution           260601 non-null int64
has_secondary_use_school                260601 non-null int64
has_secondary_use_industry              260601 non-null int64
has_secondary_use_health_post           260601 non-null int64
has_secondary_use_gov_office            260601 non-null int64
has_secondary_use_use_police            260601 non-null int64
has_secondary_use_other                 260601 non-null int64
damage_grade                            260601 non-null int64
dtypes: int64(31), object(8)
```

Data Exploration and Visualization

Cleaning the data to suit the needs, performing exploratory analysis and visualizations.

Data Cleaning and Preparation

Since `building_id` is a unique and random identifier for each building in Nepal, it does not add any value for analysis and shall be removed.

```
train_values=train_values.drop('building_id',axis=1)
train_labels=train_labels.drop('building_id',axis=1)
```

Removing *building_id*

```
all_columns = list(train_values)
numerical_columns = ['geo_level_1_id', 'geo_level_2_id', 'geo_level_3_id', 'count_floors_pre_eq', 'age', 'area_percentage', '
categorical_columns=['land_surface_condition', 'foundation_type', 'roof_type', 'ground_floor_type', 'other_floor_type', 'posi

binary_columns=[col for col in all_columns if col not in numerical_columns]
binary_columns=[col for col in binary_columns if col not in categorical_columns]

combined_train[categorical_columns] = combined_train[categorical_columns].astype('category')

combined_train[binary_columns] = combined_train[binary_columns].astype('bool')
```

Extracting the different columns and dummy encoding them to the desired type

Data Exploration and Visualization

Building features

- 260, 601 building data entries with 3 mixed feature types
  - 8 continuous (excluding building_id)
  - 8 categorical
  - 22 binary

Continuous Features

geo_level_1_id	int64
geo_level_2_id	int64
geo_level_3_id	int64
count_floors_pre_eq	int64
age	int64
area_percentage	int64
height_percentage	int64
count_families	int64

dtype: object

Categorical Features

land_surface_condition	category
foundation_type	category
roof_type	category
ground_floor_type	category
other_floor_type	category
position	category
plan_configuration	category
legal_ownership_status	category

dtype: object

Binary Features

has_superstructure_adobe_mud	bool
has_superstructure_mud_mortar_stone	bool
has_superstructure_stone_flag	bool
has_superstructure_cement_mortar_stone	bool
has_superstructure_mud_mortar_brick	bool
has_superstructure_cement_mortar_brick	bool
has_superstructure_timber	bool
has_superstructure_bamboo	bool
has_superstructure_rc_non_engineered	bool
has_superstructure_rc_engineered	bool
has_superstructure_other	bool
has_secondary_use	bool
has_secondary_use_agriculture	bool
has_secondary_use_hotel	bool
has_secondary_use_rental	bool
has_secondary_use_institution	bool
has_secondary_use_school	bool
has_secondary_use_industry	bool
has_secondary_use_health_post	bool
has_secondary_use_gov_office	bool
has_secondary_use_use_police	bool
has_secondary_use_other	bool

dtype: object

Data Exploration and Visualization

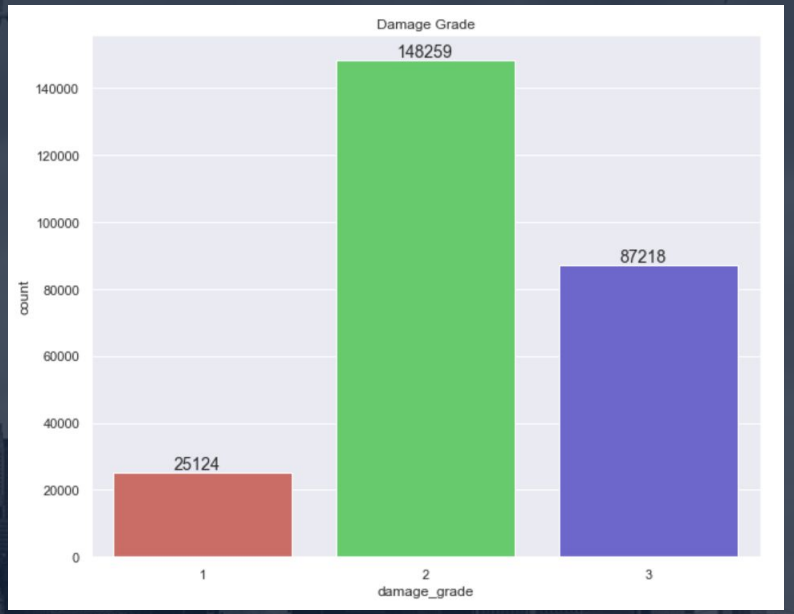
Response variable

- 260, 601 building data entries with ordinal type
- 3 *ordinal* classes categorisation:
  - 1 - Low Damage
  - 2 - Medium Damage
  - 3 - High Damage

```
count      260601
unique          3
top             2
freq       148259
Name: damage_grade, dtype: int64
```

Data Exploration and Visualization

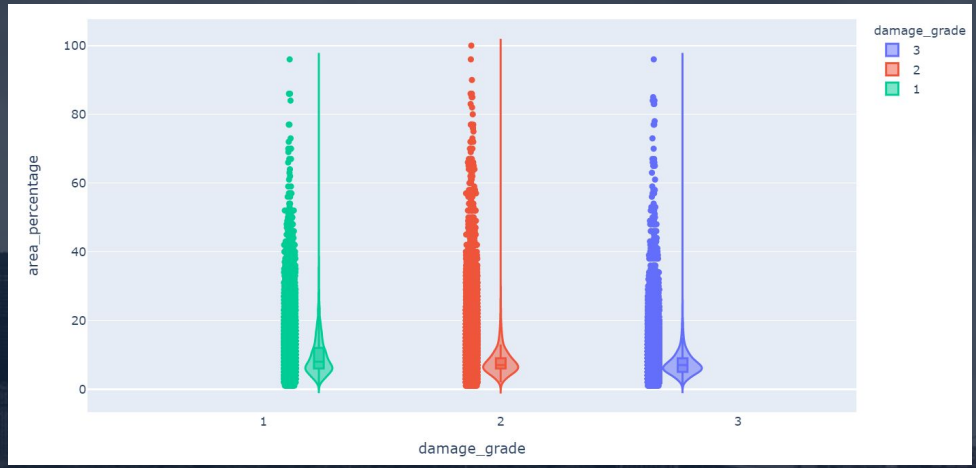
- Majority of the buildings suffered medium damage
- 3 classes are imbalanced



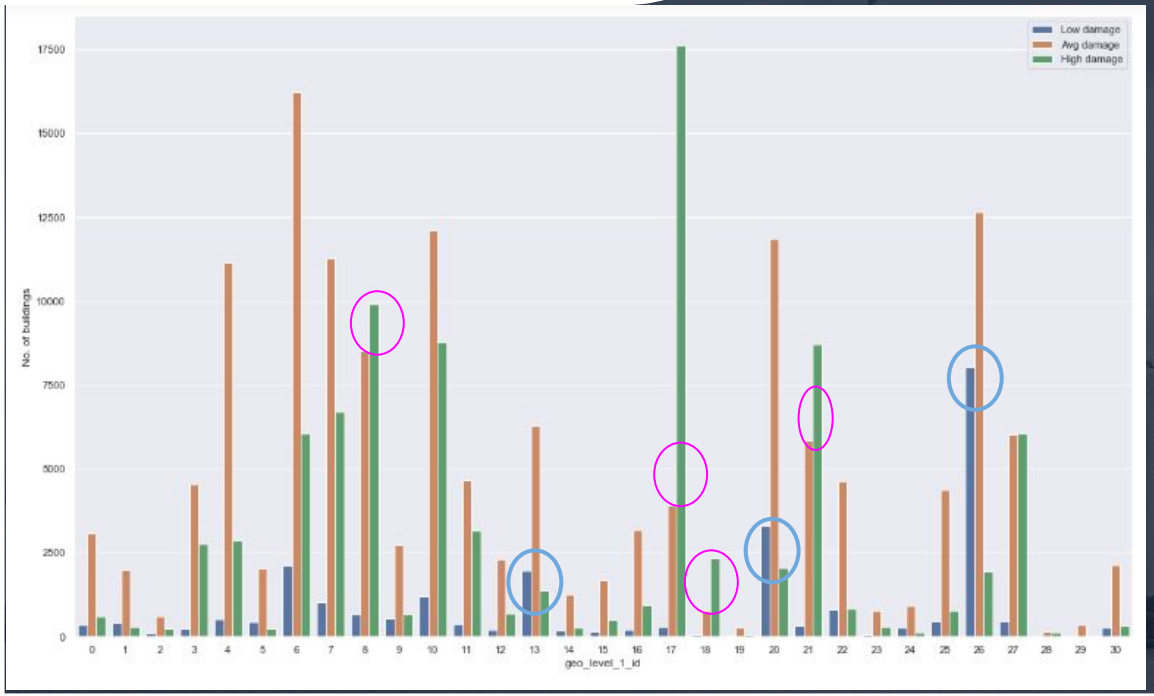
Data Exploration and Visualization

Plotting features vs damage_grade

- Continuous



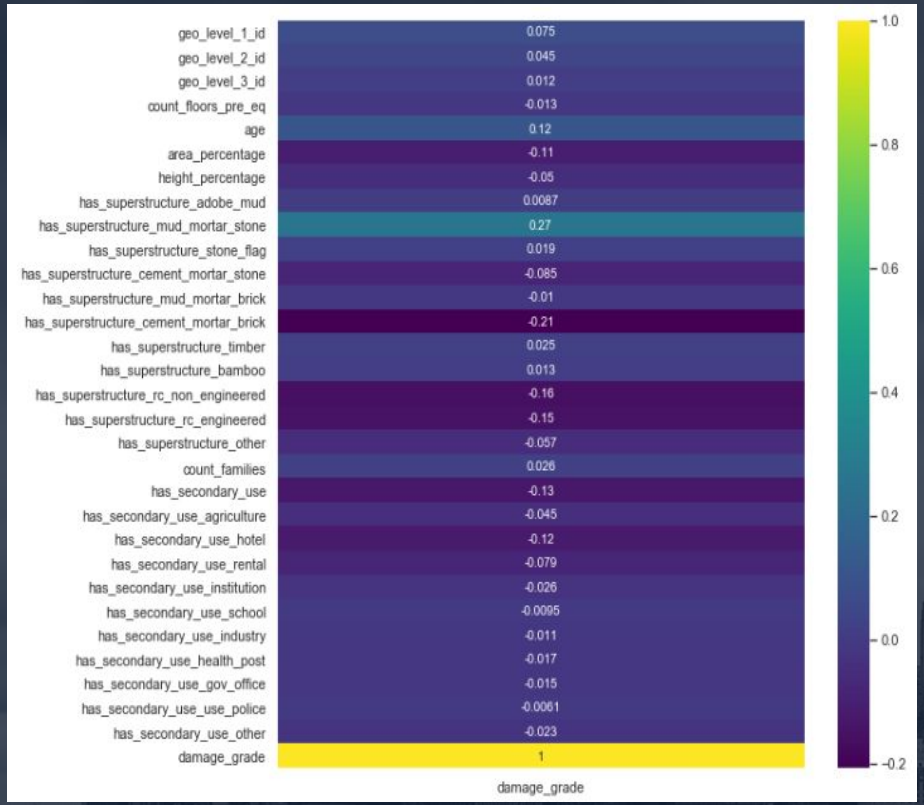
geo_level_1_id vs damage_grade



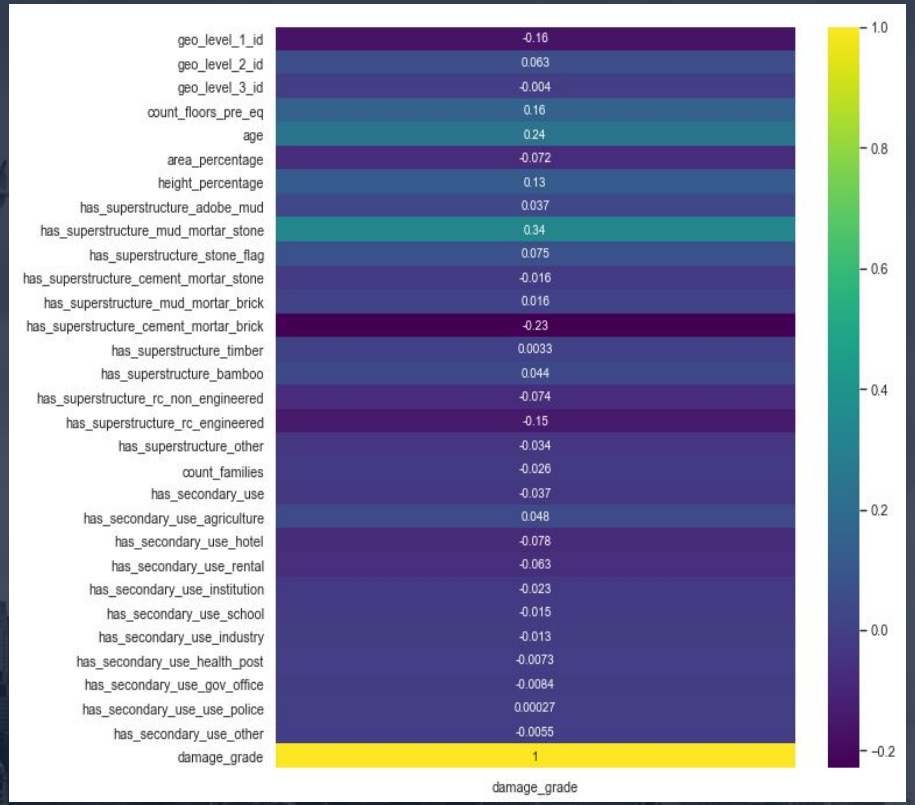
geo_level_1_id values 8, 17, 18, 21 have more higher-damaged buildings

geo_level_1_id values 13, 20, 26 have more lower-damaged buildings

Higher-damaged



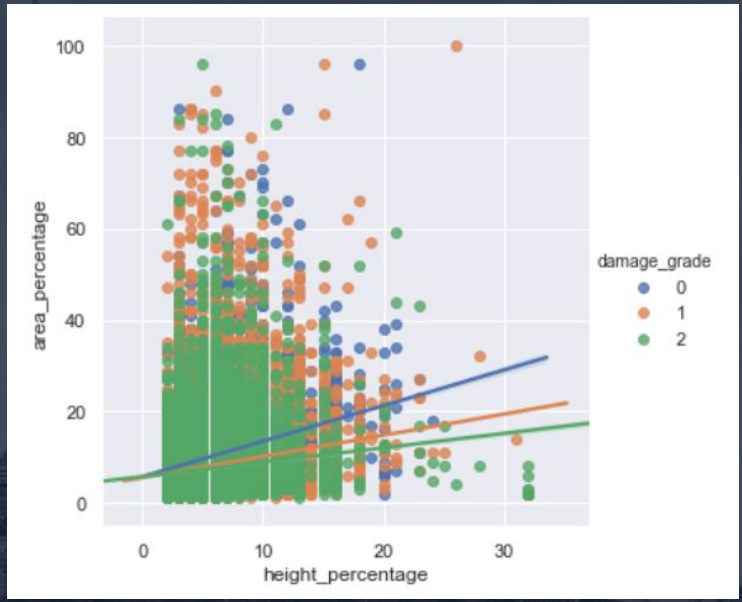
Lower-damaged



Continuous features: age, count_floors_pre_eq, area_percentage
Binary features: has_superstructure_mud_mortar_stone, has_superstructure_cement_mortar_brick, has_superstructure_rc_engineered

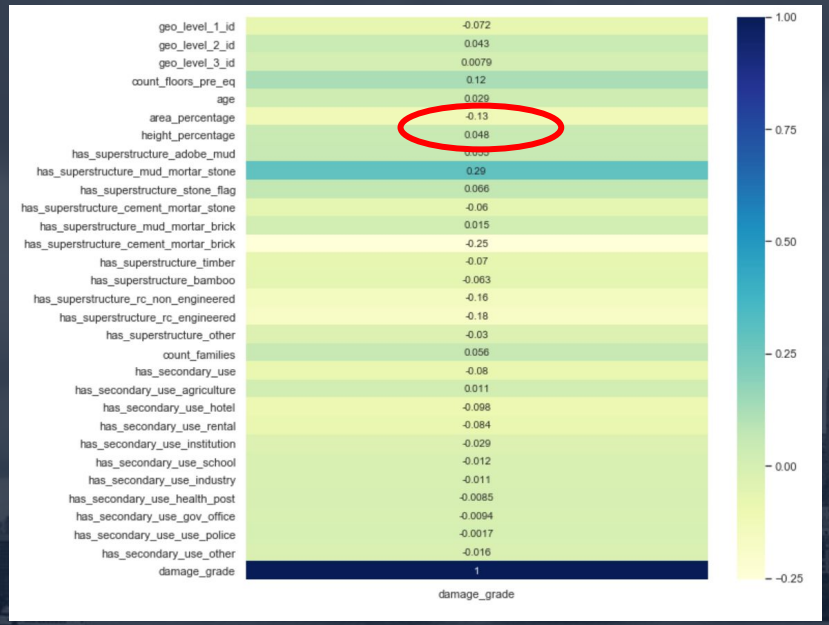
An Interesting Observation

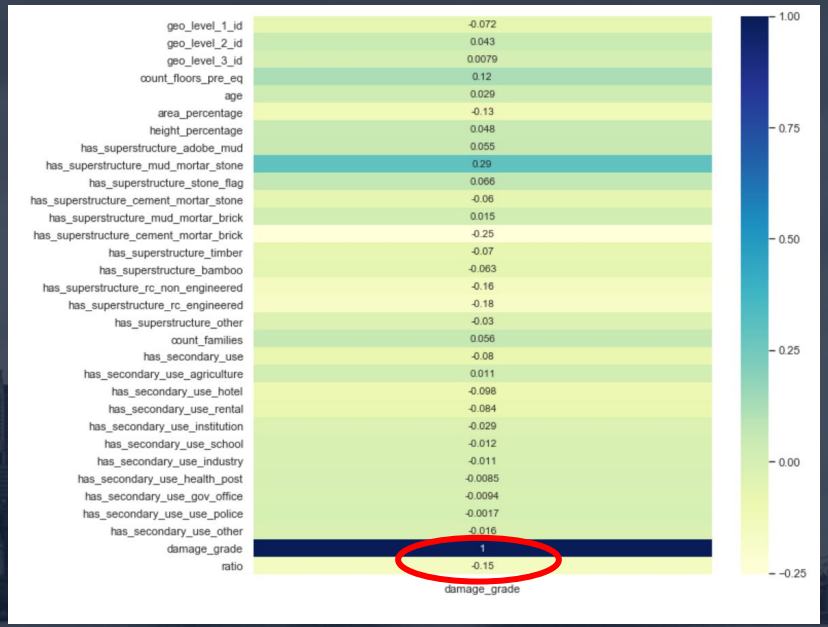
Plotting area_percentage against height_percentage



An Interesting Observation

Introducing a new variable
Ratio=area_percentage/height_percentage

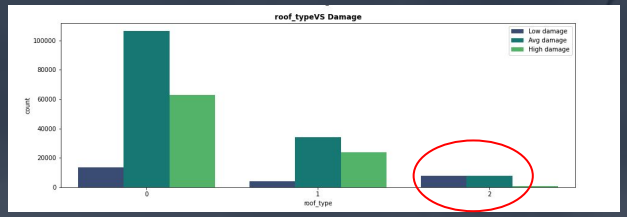


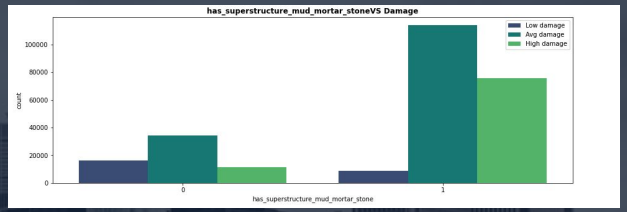


Data Exploration and Visualization

Variations in proportion for each feature

- Categorical
- Boolean





Statistical Analysis

Selecting the most important features that affect the damage.

Statistical Analysis

Top 3 continuous features (out of 8)

```
numerical_x_train = train_values[numerical_columns]
selector = SelectKBest(f_classif, 3).fit(numerical_x_train, train_labels.values.ravel())
cols = selector.get_support(indices=True)
features_df_new = numerical_x_train.iloc[:,cols]
best_num = list(features_df_new)
print(best_num)
```

```
['geo_level_1_id', 'count_floors_pre_eq', 'area_percentage']
```

Top 3 categorical features (out of 8)

```
categorical_x_train = train_values[categorical_columns]
selector = SelectKBest(mutual_info_classif, 3).fit(categorical_x_train, train_labels.values.ravel())
cols = selector.get_support(indices=True)
features_df_new = categorical_x_train.iloc[:,cols]
best_cat = list(features_df_new)
print(best_cat)
```

```
['foundation_type', 'ground_floor_type', 'other_floor_type']
```

Top 5 binary features (out of 22)

```
binary_x_train = train_values[binary_columns]
selector = SelectKBest(mutual_info_classif, 5).fit(binary_x_train, train_labels.values.ravel())
cols = selector.get_support(indices=True)
features_df_new = binary_x_train.iloc[:,cols]
best_bin = list(features_df_new)
print(best_bin)
```

```
['has_superstructure_mud_mortar_stone', 'has_superstructure_cement_mortar_brick', 'has_superstructure
_rc_non_engineered', 'has_superstructure_rc_engineered', 'has_secondary_use']
```

Feature selection

- Continuous features
  - F_test
- Categorical/Boolean features
  - Mutual Information

Machine Learning

Training models and predicting the damage grade of buildings.

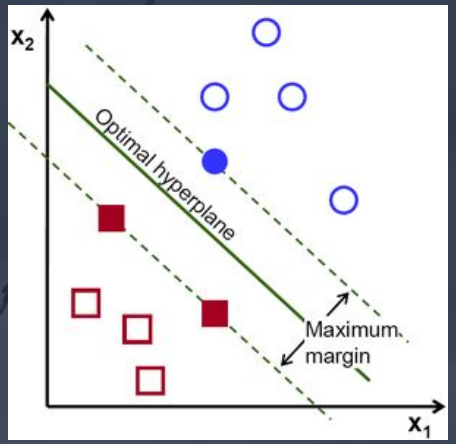
Machine Learning
GridSearchCV

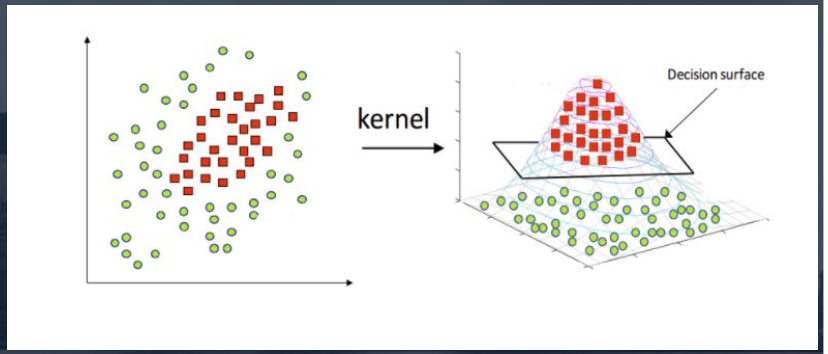
```
In [ ]:
1 grid = GridSearchCV(estimator, param_grid, refit=True, verbose=3, cv=5, n_jobs=4)
2 grid.fit(X_train, Y_train)
3 grid_predictions = grid.predict(X_test)
4
5
6 print("Accuracy is: ", grid.score(X_test, Y_test))
7 print("The best params are: " + str(grid.best_params_))
8 print("The best estimator is: " + str(grid.best_estimator_))
9 print(str(classification_report(Y_test, grid_predictions)))
```

- Optimise results by selecting hyperparameters from pre-defined “grid” of values
- Cross Validation to ensure reliable results

Machine Learning
Support Vector Machine (SVM)

- Supervised ML technique
- Parameters
  - C, Kernel, Gamma
- Data Preparation
  - One-Hot Encoding






```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV] C=1 ..............................................................
/home/students/student2_17/anaconda3/envs/cy2001/lib/python3.5/site-packages/sklea
  y = column_or_1d(y, warn=True)
/home/students/student2_17/anaconda3/envs/cy2001/lib/python3.5/site-packages/sklea
'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
ad[CV] ................. C=1, score=0.7136895625479662, total=454.8min
```

Machine Learning
Predicting damage grade

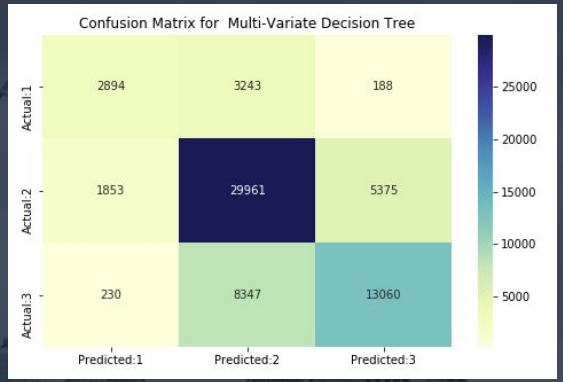
Multivariate Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

X_train, X_test, y_train, y_test = train_test_split(train_values, train_labels, test_size = 0.25)

dectree = DecisionTreeClassifier(max_depth = 16)
dectree.fit(X_train, y_train)

y_train_pred = dectree.predict(X_train)
y_test_pred = dectree.predict(X_test)

print("Classification Accuracy :", dectree.score(X_test, y_test))
```

```
Classification Accuracy : 0.7047474328866786
```

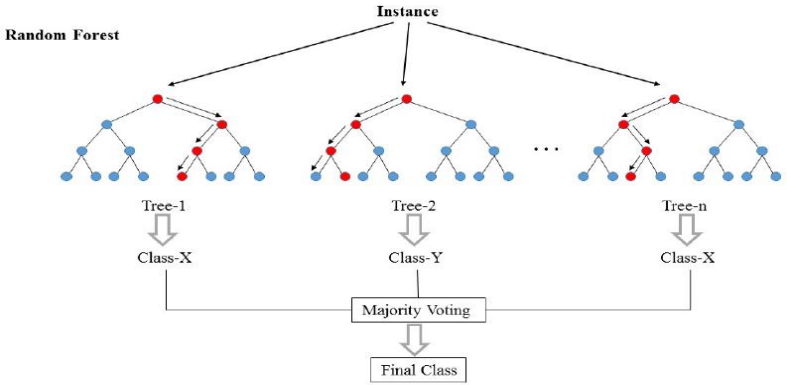
```
print("Classification Report: ", classification_report(y_test, y_test_pred))
```

```
Classification Report:               precision    recall  f1-score   support

           1       0.58      0.46      0.51      6325
           2       0.72      0.81      0.76     37189
           3       0.70      0.60      0.65     21637

    accuracy                           0.70     65151
   macro avg       0.67      0.62      0.64     65151
weighted avg       0.70      0.70      0.70     65151
```



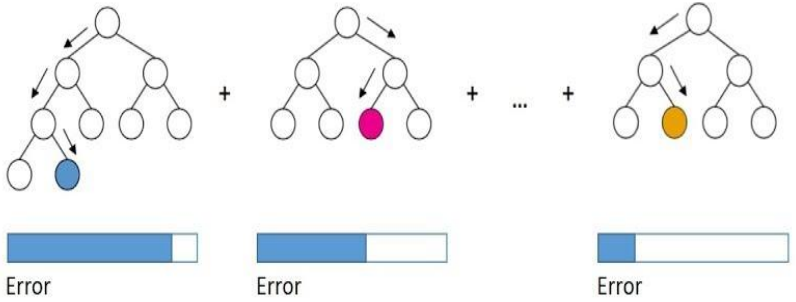
Machine Learning
Ensemble methods

Random Forest Classifier



An accuracy of ~72% is achieved.

Gradient Boosting Classifier



An accuracy of ~68% is achieved from GradientBoosting and ~74% from XGBClassifier (with GridSearchCV).

[1] https://tex.stackexchange.com/questions/503883/illustrating-the-random-forest-algorithm-in-tikz
[2] https://sefiks.com/2018/10/04/a-step-by-step-gradient-boosting-decision-tree-example/

Machine Learning
Random Forest

```
rf = RandomForestClassifier(n_estimators = 150) # with 150 trees
rf.fit(X_train,Y_train)
rf_pred=rf.predict(X_test)
# print the accuracy
print("Accuracy:",metrics.accuracy_score(Y_test, rf_pred))
print(" \n")
```

```
Accuracy: 0.7207459565242417
```

```
print("classification report for Random Forest :\n\n",metrics.classification_report(Y_test, rf_pred))
```

```
classification report for Random Forest :

              precision    recall  f1-score   support

           0       0.65      0.48      0.56      5023
           1       0.73      0.82      0.77     29681
           2       0.72      0.61      0.66     17417

    accuracy                           0.72     52121
   macro avg       0.70      0.64      0.66     52121
weighted avg       0.72      0.72      0.72     52121
```

Confusion Matrix

n_estimators: Number of trees in the forest

Machine Learning
Gradient Boosting

```
GBC=GradientBoostingClassifier(n_estimators=100)
GBC.fit(X_train,Y_train)
GBC_pred=GBC.predict(X_test)
print("Accuracy:",metrics.accuracy_score(Y_test, GBC_pred))
print(" \n")
print("classification report for Gradient Boosting Classifier :\n\n",metrics.classification_report(Y_test, GBC_pred))
```

```
Accuracy: 0.683275455190806


classification report for Random Forest :

              precision    recall  f1-score   support

           0       0.63      0.37      0.47      5023
           1       0.68      0.86      0.76     29681
           2       0.72      0.47      0.57     17417

    accuracy                           0.68     52121
   macro avg       0.68      0.57      0.60     52121
weighted avg       0.69      0.68      0.67     52121
```

Machine Learning
XGBClassifier

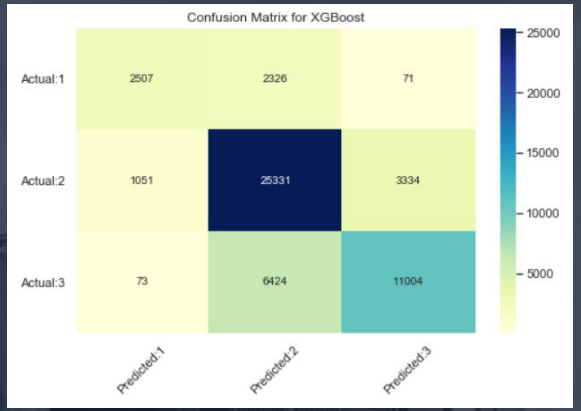
- ‘Regularized boosting‘ technique
- Implements parallel processing

```
XGB.fit(X_train, y_train.values.ravel())
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='multi:softprob', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

```
XGB_y_pred=XGB.predict(X_test)
print("Accuracy:", metrics.accuracy_score(y_test, XGB_y_pred))
print(" \n")
print("Classification report for XGBClassifier:\n\n", metrics.classification_r
```

```
Accuracy: 0.674027743136164


Classification report for XGBClassifier:

              precision    recall  f1-score   support

           1       0.63      0.35      0.45      5162
           2       0.66      0.88      0.75     29511
           3       0.73      0.43      0.54     17448

    accuracy                           0.67     52121
   macro avg       0.67      0.55      0.58     52121
weighted avg       0.68      0.67      0.65     52121
```



Machine Learning GridSearchCV example

- After fine-tuning of hyperparameters from the grid shown
  => Accuracy score improved

Before: 0.674027743136164

After: 0.744172102839601

```
model = XGBClassifier()

parameters = {'nthread':[4],
              'objective':['binary:logistic'],
              'learning_rate': [0.05, 0.1],
              'max_depth': [2,4,6],
              'min_child_weight': [3,6,11],
              'silent': [1],
              'subsample': [0.8],
              'colsample_bytree': [0.3, 0.7],
              'n_estimators': [100,500,1000],
              'missing':[-999],
              'seed': [1337]}


clf = GridSearchCV(model, parameters, n_jobs=5,
                   cv=StratifiedKFold(n_splits=5, shuffle=True),
                   verbose=2, refit=True)
```

```
clf.fit(x_train_new,y_train)
print(clf.best_score_)
print(clf.best_params_)
```

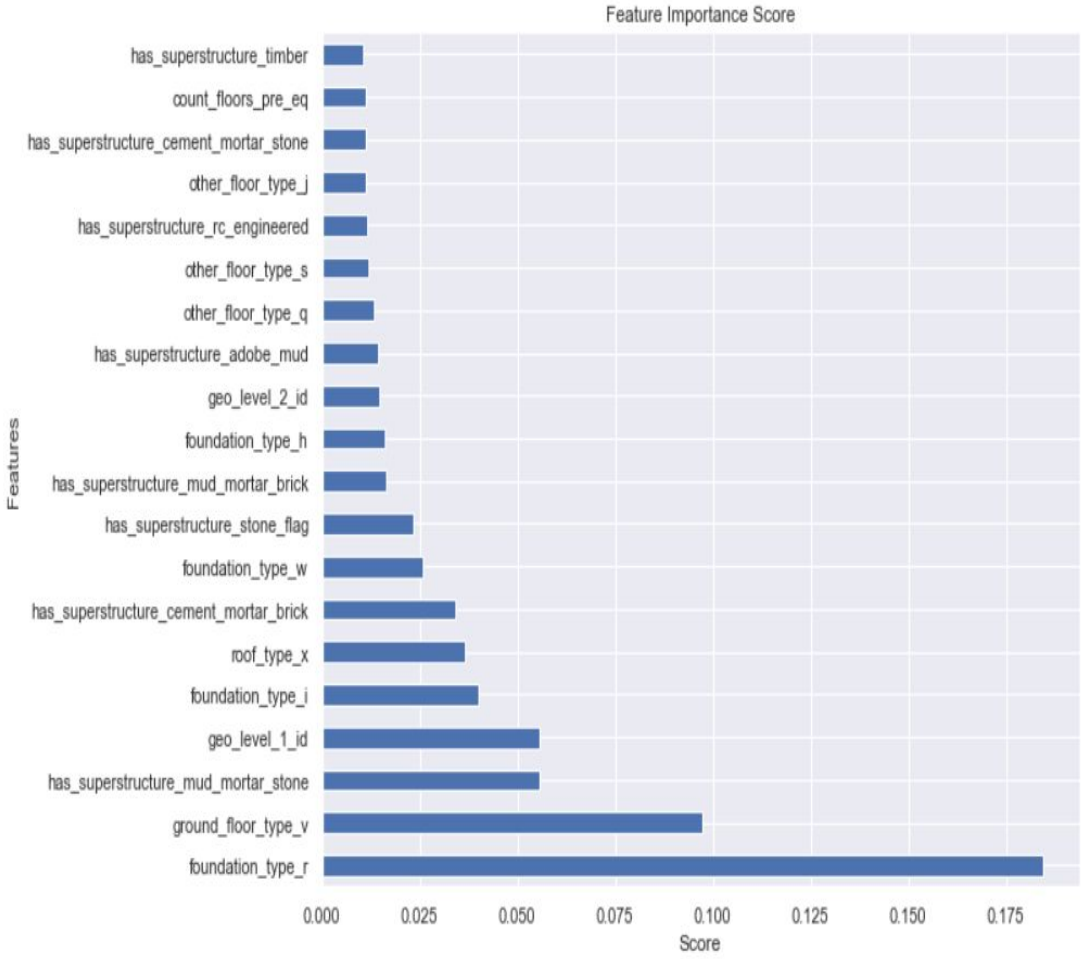
```
Fitting 5 folds for each of 108 candidates, totalling 540 fits

[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done  31 tasks       | elapsed: 23.1min
[Parallel(n_jobs=5)]: Done 152 tasks       | elapsed: 169.8min
[Parallel(n_jobs=5)]: Done 355 tasks       | elapsed: 413.1min
[Parallel(n_jobs=5)]: Done 540 out of 540 | elapsed: 911.6min finished
```

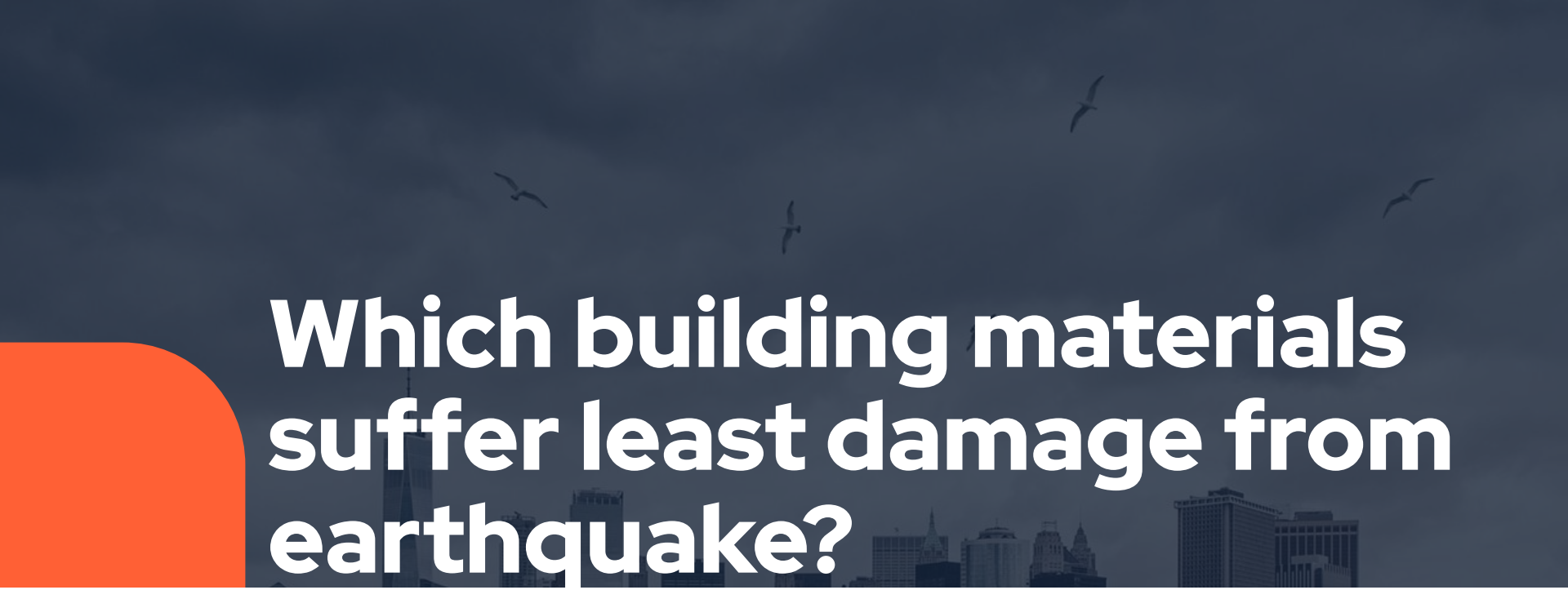
Optimal parameters:

```
{'colsample_bytree': 0.7, 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 6, 'missing': -999, 'n_estimators': 1000,
'nthread': 4, 'objective': 'binary:logistic', 'seed': 1337, 'silent': 1, 'subsample': 0.8}
```

Feature Importance

Top 20 factors (in categories) affecting damage grade (in ascending order from top to bottom) using model with highest accuracy score (XGBClassifier)





Which building materials suffer least damage from earthquake?

Choosing the ideal materials that sustain least damage.

Data Exploration
Ideal Materials

```
df=pd.read_csv('train_set.csv')
lab=pd.read_csv('train_labels.csv')
df['damage_grade']=lab['damage_grade']
for feat in categorical_columns:
    grouped = df.groupby(feat)
    print(grouped['damage_grade'].agg([np.sum, np.mean, np.std]))
```

The ‘ideal’ materials identified are -

land_surface_condition = t

foundation_type = i

roof_type = x

ground_floor_type = x

other_floor_type = x

has_superstructure = cement_mortar_brick and rc_engineered

Data Exploration
Ideal Materials

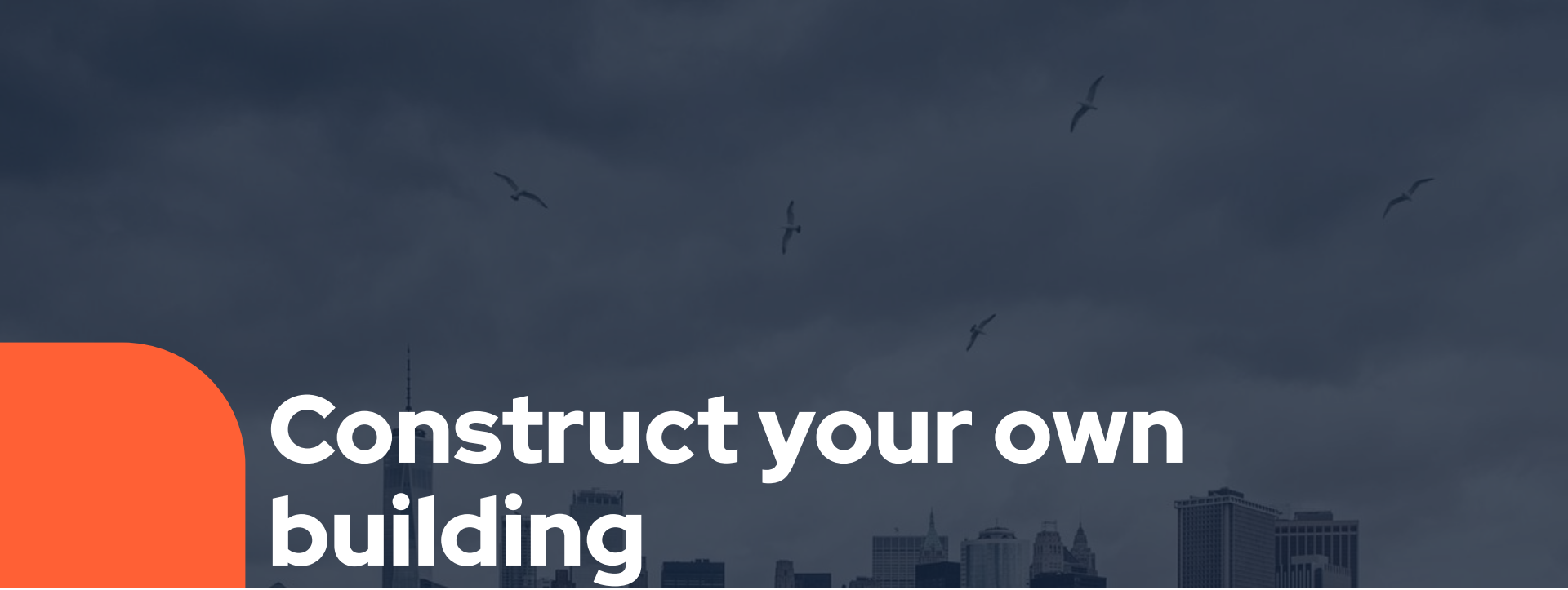
```
X_train, X_test,Y_train, Y_test = train_test_split(train,damage_grade,test_size = 0.20)
model = XGBClassifier(max_depth=9,min_child_weight=1,nthread=4)
model.fit(X_train,Y_train)
pred=model.predict(X_test)
```

Training the dataset using only the columns related to materials.

```
pred_ideal=model.predict(ideal)
pred_ideal
```

```
array([1], dtype=int64)
```

Predicting the damage grade of the building with the ideal materials irrespective of the location using the model trained earlier.
The predicted damage grade comes out to be 1 (Low Damage).



Construct your own building

Asking the user to input materials being used in the construction to predict the damage grade.

Construct your own building

```
print("Let us predict the damage grade of your building in case of earthquake \n")
print("Enter the details according to the instruction provided \n")
a=input("Enter land surface condition- n,o,t \n")
b=input("Enter type of foundation- h,i,r,u,w \n")
c=input("Enter roof type -n,q,x \n")
d=input("Enter ground floor type -f,m,v,x,z \n")
e=input("Enter other floor type -j,q,s,x \n")
f=input("Enter type of superstructure - 0 for adobe_mud, 1 for mud_mortar_stone, 2 for stone_flag, \
         3 for cement_mortar_stone, 4 for mud_mortar_brick, 5 for cement_mortar_brick, \
         6 for timber, 7 for bamboo and 8 for others \n")
g=input("Is the superstructure rc engineered? 0 for False, 1 for True")
```

Taking input from the user regarding the various building materials being used.

```
pred_damage
```

```
array([2], dtype=int64)
```

Predicted damage grade by using the XGBoost model trained earlier.

Conclusion

What are the most important factors which affect the damage grade of the buildings?	Predict the damage grade of the buildings by training the models based on the given dataset.	What are the ideal materials to be used in the construction that would suffer least damage?	Create your own building. Predict the damage it might suffer in case of earthquake.
In descending order: - Foundation type = r - Ground floor type = v - Has superstructure mud mortar stone - Geo level 1 id - Foundation type = i - Roof type = x - Has superstructure cement mortar brick - Foundation type = w - Has superstructure stone flag - Has superstructure cement mortar brick	**Using our trained XGB Classifier model, we have predicted the damage grade of buildings listed in the test_values.csv file.**	The 'ideal' materials: - Land surface condition = t - Foundation type = i - Roof type = x - Ground floor type = x - Other floor type = x - superstructure = cement mortar brick and rc engineered	**We allowed users to input various building materials and predicted damage grade using our XGB Classifier trained model.**

Contributions

Chua Chiah Soon	Garg Astha	Ho Wei Yi Stanley	Nurul Afiqah Binte Aris
• Problem Statements • Data Cleaning and Preparation • Statistical Analysis • Machine Learning (GridSearchCV, SVM and Decision Tree) • Presentation	• Problem Statements • Data Exploration and Visualization • Machine Learning (Decision Tree, Random Forest and Gradient Boosting) • Identifying ideal building materials • Construct your own building • Presentation	• Problem Statements • Data Cleaning and Preparation • Data Exploration and Visualization • Machine Learning (XGBoost and GridSearchCV) • Presentation	• Introduction • Problem Statements • Description of Dataset • Conclusion • Presentation


Thank You!
Any questions?