

4.5. Earthquake Damage in Kavrepalanchok NP

In this assignment, you'll build a classification model to predict building damage for the district of [Kavrepalanchok](#).

```
import warnings  
  
import wqet_grader  
  
warnings.simplefilter(action="ignore", category=FutureWarning)  
wqet_grader.init("Project 4 Assessment")
```

```
# Import Libraries here  
import sqlite3  
import warnings  
  
import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
import seaborn as sns  
from category_encoders import OneHotEncoder, OrdinalEncoder  
from IPython.display import VimeoVideo  
from sklearn.linear_model import LogisticRegression  
from sklearn.metrics import accuracy_score  
from sklearn.model_selection import train_test_split  
from sklearn.pipeline import Pipeline, make_pipeline  
from sklearn.utils.validation import check_is_fitted  
from sklearn.tree import DecisionTreeClassifier
```

Prepare Data

Connect

Run the cell below to connect to the `nepal.sqlite` database.

```
%load_ext sql  
%sql sqlite:///home/jovyan/nepal.sqlite
```

```
: %sql sqlite:///home/jovyan/nepal.sqlite
```

There's a new jupysql version available (0.10.10), you're running 0.10.1. To upgrade: pip install jupysql --upgrade

Warning: Be careful with your SQL queries in this assignment. If you try to get all the rows from a table (for example, `SELECT * FROM id_map`), you will cause an Out of Memory error on your virtual machine. So always include a `LIMIT` when first exploring a database.

Task 4.5.1: What districts are represented in the `id_map` table? Determine the unique values in the `district_id` column.

```
: %%sql  
  
SELECT distinct(district_id)  
FROM id_map
```

Running query in 'sqlite:///home/jovyan/nepal.sqlite'

```
: district_id  
1  
2  
3  
4
```

```
: result = _.DataFrame().squeeze() # noqa F821  
wget_grader.grade("Project 4 Assessment", "Task 4.5.1", result)
```

That's the right answer. Keep it up!

Score: 1

What's the district ID for Kavrepalanchok? From the lessons, you already know that Gorkha is 4; from the textbook, you know that Ramechhap is 2. Of the remaining districts, Kavrepalanchok is the one with the largest number of observations in the `id_map` table.

Task 4.5.2: Calculate the number of observations in the `id_map` table associated with district 1.

```
: %%sql  
  
SELECT count(*)  
FROM id_map
```

```
SELECT count(*)
FROM id_map
WHERE district_id = 1
```

Running query in 'sqlite:///home/jovyan/nepal.sqlite'

```
: count(*)
```

36112

```
: result = [_.DataFrame().astype(float).squeeze()] # noqa F821
wqet_grader.grade("Project 4 Assessment", "Task 4.5.2", result)
```

You're making this look easy. 😊

Score: 1

Task 4.5.3: Calculate the number of observations in the `id_map` table associated with district `3`.

```
%%sql
SELECT count(*)
FROM id_map
WHERE district_id = 3
```

Running query in 'sqlite:///home/jovyan/nepal.sqlite'

```
: count(*)
```

82684

```
: result = [_.DataFrame().astype(float).squeeze()] # noqa F821
wqet_grader.grade("Project 4 Assessment", "Task 4.5.3", result)
```

That's the right answer. Keep it up!

Score: 1

Task 4.5.4: Join the unique building IDs from Kavrepalanchok in `id_map`, all the columns from `building_structure`, and the `damage_grade` column from `building_damage`, limiting your results to 5 rows. Make sure you rename the `building_id` column in `id_map` as `b_id` and limit your results to the first five rows of the new table.

```
: %%sql
```

Task 4.5.4: Join the unique building IDs from Kavrepalanchok in `id_map`, all the columns from `building_structure`, and the `damage_grade` column from `building_damage`, limiting your results to 5 rows. Make sure you rename the `building_id` column in `id_map` as `b_id` and limit your results to the first five rows of the new table.

```
: %%sql
```

```
SELECT distinct(i.building_id) AS b_id, s.* , d.damage_grade
FROM id_map AS i
JOIN building_structure AS s ON i.building_id = s.building_id
JOIN building_damage AS d ON i.building_id = d.building_id
WHERE district_id=3
LIMIT 5
```

Running query in 'sqlite:///home/jovyan/nepal.sqlite'

b_id	building_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
87473	87473	2	1	15	382	18	7	Flat
87479	87479	1	0	12	328	7	0	Flat
87482	87482	2	1	23	427	20	7	Flat
87491	87491	2	1	12	427	14	7	Flat
87496	87496	2	0	32	360	18	0	Flat

```
: result = _.DataFrame().set_index("b_id") # noqa F821
wqet_grader.grade("Project 4 Assessment", "Task 4.5.4", result)
```

That's the right answer. Keep it up!

Score: 1

Import

Task 4.5.5: Write a `wrangle` function that will use the query you created in the previous task to create a DataFrame. In addition your function should:

1. Create a "severe_damage" column, where all buildings with a damage grade greater than `3` should be encoded as `1`. All other buildings should be encoded at `0`.
2. Drop any columns that could cause issues with leakage or multicollinearity in your model.

```
|: # Build your `wrangle` function here
def wrangle(db_path):
    # Connect to database
    conn = sqlite3.connect(db_path)

    # Construct query
    query = """
        SELECT distinct(i.building_id) AS b_id, s.* , d.damage_grade
        FROM id_map AS i
        JOIN building_structure AS s ON i.building_id = s.building_id
        JOIN building_damage AS d ON i.building_id = d.building_id
        WHERE district_id=3
        """

    # Read query results into DataFrame
    df = pd.read_sql(query, conn, index_col="b_id")

    # Identify Leaky columns
    drop_cols=[col for col in df.columns if "post_eq" in col]

    # Create binary target
    df["damage_grade"] = df["damage_grade"].str[-1].astype(int)
    df["severe_damage"] = (df["damage_grade"] > 3).astype(int)

    drop_cols.extend(("damage_grade", "count_floors_pre_eq", "building_id"))
    # Drop columns
    df.drop(columns=drop_cols, inplace=True)

    return df
```

Use your `wrangle` function to query the database at `"/home/jovyan/nepal.sqlite"` and return your cleaned results.

```
|: df = wrangle("/home/jovyan/nepal.sqlite")
df.head()
```

```
df = wrangle("/home/jovyan/nepal.sqlite")
df.head()
```

b_id	age_building	plinth_area_sq_ft	height_ft_pre_eq	land_surface_condition	foundation_type	roof_type	ground_floor_type	other_floor_type	p
87473	15	382	18	Flat	Mud mortar-Stone/Brick	Bamboo/Timber-Light roof	Mud	Timber/Bamboo-Mud	at
87479	12	328	7	Flat	Mud mortar-Stone/Brick	Bamboo/Timber-Light roof	Mud	Not applicable	at
87482	23	427	20	Flat	Mud mortar-Stone/Brick	Bamboo/Timber-Light roof	Mud	Timber/Bamboo-Mud	at
87491	12	427	14	Flat	Mud mortar-Stone/Brick	Bamboo/Timber-Light roof	Mud	Timber/Bamboo-Mud	at
87496	32	360	18	Flat	Mud mortar-Stone/Brick	Bamboo/Timber-Light roof	Mud	Timber/Bamboo-Mud	at

```
wqet_grader.grade(
    "Project 4 Assessment", "Task 4.5.5", wrangle("/home/jovyan/nepal.sqlite")
)
```

Way to go!

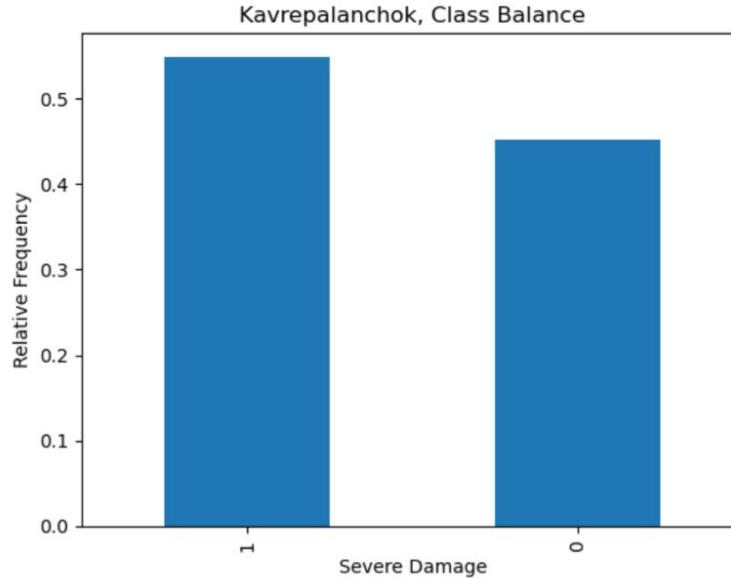
Score: 1

Explore

Task 4.5.6: Are the classes in this dataset balanced? Create a bar chart with the normalized value counts from the "severe_damage" column. Be sure to label the x-axis "Severe Damage" and the y-axis "Relative Frequency". Use the title "Kavrepalanchok, Class Balance".

```
# Plot value counts of `severe_damage`
df["severe_damage"].value_counts(normalize=True).plot(
    kind="bar", xlabel="Severe Damage", ylabel="Relative Frequency", title="Kavrepalanchok, Class Balance"
);
# Don't delete the code below 🌟
```

```
);  
# Don't delete the code below 🌟  
plt.savefig("images/4-5-6.png", dpi=150)
```



```
)]: with open("images/4-5-6.png", "rb") as file:  
    wqet_grader.grade("Project 4 Assessment", "Task 4.5.6", file)
```

Yes! Your hard work is paying off.

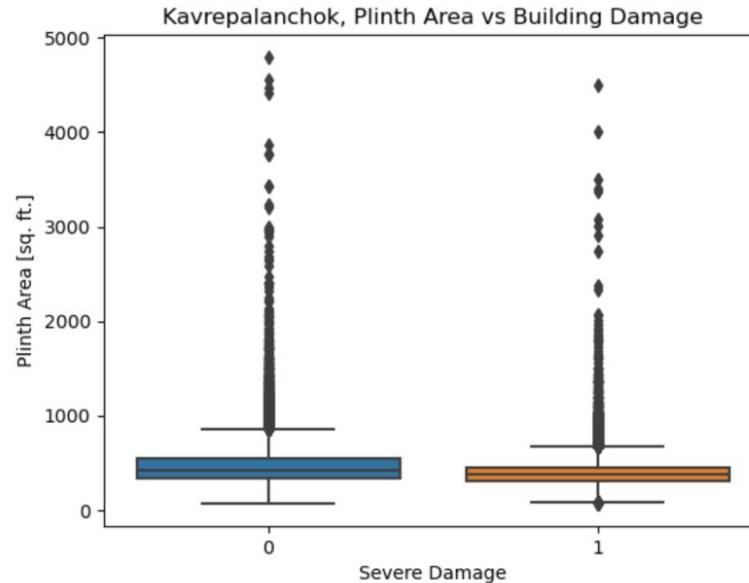
Score: 1

Task 4.5.7: Is there a relationship between the footprint size of a building and the damage it sustained in the earthquake? Use seaborn to create a boxplot that shows the distributions of the "plinth_area_sq_ft" column for both groups in the "severe_damage" column. Label your x-axis "Severe Damage" and y-axis "Plinth Area [sq. ft.]". Use the title "Kavrepalanchok, Plinth Area vs Building Damage".

```
)]: sns.boxplot(x="severe_damage", y="plinth_area_sq_ft", data=df)  
plt.xlabel("Severe Damage")
```

Task 4.5.7: Is there a relationship between the footprint size of a building and the damage it sustained in the earthquake? Use seaborn to create a boxplot that shows the distributions of the "plinth_area_sq_ft" column for both groups in the "severe_damage" column. Label your x-axis "Severe Damage" and y-axis "Plinth Area [sq. ft.]". Use the title "Kavrepalanchok, Plinth Area vs Building Damage".

```
sns.boxplot(x="severe_damage", y="plinth_area_sq_ft", data=df)
plt.xlabel("Severe Damage")
plt.ylabel("Plinth Area [sq. ft.]")
plt.title("Kavrepalanchok, Plinth Area vs Building Damage");
# Don't delete the code below 👇
plt.savefig("images/4-5-7.png", dpi=150)
```



```
with open("images/4-5-7.png", "rb") as file:
    wget_grader.grade("Project 4 Assessment", "Task 4.5.7", file)
```

Boom! You got it.

Score: 1

Task 4.5.8: Are buildings with certain roof types more likely to suffer severe damage? Create a pivot table of `df` where the index is `"roof_type"` and the values come from the `"severe_damage"` column, aggregated by the mean.

```
]: roof_pivot = pd.pivot_table(  
    df, index="roof_type", values="severe_damage", aggfunc=np.mean  
)  
roof_pivot
```

```
]:
```

severe_damage	roof_type
0.569477	Bamboo/Timber-Heavy roof
0.604842	Bamboo/Timber-Light roof
0.040715	RCC/RB/RBC

```
]: wqet_grader.grade("Project 4 Assessment", "Task 4.5.8", roof_pivot)
```

Correct.

Score: 1

Split

Task 4.5.9: Create your feature matrix `X` and target vector `y`. Your target is `"severe_damage"`.

```
]: target = "severe_damage"  
X = df.drop(columns=target)  
y = df[target]  
print("X shape:", X.shape)  
print("y shape:", y.shape)
```

```
X shape: (76533, 11)  
y shape: (76533,)
```

```
]: wqet_grader.grade("Project 4 Assessment", "Task 4.5.9a", X)
```

```
: wqet_grader.grade("Project 4 Assessment", "Task 4.5.9a", X)
```

Excellent work.

Score: 1

```
: wqet_grader.grade("Project 4 Assessment", "Task 4.5.9b", y)
```

Excellent! Keep going.

Score: 1

Task 4.5.10: Divide your dataset into training and validation sets using a randomized split. Your validation set should be 20% of your data.

```
: X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_val shape:", X_val.shape)
print("y_val shape:", y_val.shape)
```

```
X_train shape: (61226, 11)
y_train shape: (61226,)
X_val shape: (15307, 11)
y_val shape: (15307,)
```

```
: wqet_grader.grade("Project 4 Assessment", "Task 4.5.10", [X_train.shape == (61226, 11)])
```

Awesome work.

Score: 1

Build Model

Baseline

Task 4.5.11: Calculate the baseline accuracy score for your model.

Task 4.5.11: Calculate the baseline accuracy score for your model.

```
|: acc_baseline = y_train.value_counts(normalize=True).max()
|: print("Baseline Accuracy:", round(acc_baseline, 2))
Baseline Accuracy: 0.55
|: wqet_grader.grade("Project 4 Assessment", "Task 4.5.11", [acc_baseline])
```

Very impressive.

Score: 1

Iterate

Task 4.5.12: Create a model `model_lr` that uses logistic regression to predict building damage. Be sure to include an appropriate encoder for categorical features.

```
|: model_lr = make_pipeline(
|:     OneHotEncoder(use_cat_names=True),
|:     LogisticRegression(max_iter=1000)
|: )
|: model_lr.fit(X_train, y_train)
Pipeline(steps=[('onehotencoder',
|:     OneHotEncoder(cols=['land_surface_condition',
|:                         'foundation_type', 'roof_type',
|:                         'ground_floor_type', 'other_floor_type',
|:                         'position', 'plan_configuration',
|:                         'superstructure'],
|:                         use_cat_names=True)),
|:     ('logisticregression', LogisticRegression(max_iter=1000))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
|: wqet_grader.grade("Project 4 Assessment", "Task 4.5.12", model_lr)
```

Good work!

Score: 1

Good work!

Score: 1

Task 4.5.13: Calculate training and validation accuracy score for `model_lr`.

```
: lr_train_acc = model_lr.score(X_train, y_train)
lr_val_acc = model_lr.score(X_val, y_val)

print("Logistic Regression, Training Accuracy Score:", lr_train_acc)
print("Logistic Regression, Validation Accuracy Score:", lr_val_acc)
```

```
Logistic Regression, Training Accuracy Score: 0.6513735994512135
Logistic Regression, Validation Accuracy Score: 0.6522506042986869
```

```
: submission = [lr_train_acc, lr_val_acc]
wget_grader.grade("Project 4 Assessment", "Task 4.5.13", submission)
```

Awesome work.

Score: 1

Task 4.5.14: Perhaps a decision tree model will perform better than logistic regression, but what's the best hyperparameter value for `max_depth`? Create a `for` loop to train and evaluate the model `model_dt` at all depths from 1 to 15. Be sure to use an appropriate encoder for your model, and to record its training and validation accuracy scores at every depth. The grader will evaluate your validation accuracy scores only.

```
: depth_hyperparams = range(1, 16)
training_acc = []
validation_acc = []

for d in depth_hyperparams:
    model_dt = make_pipeline(
        OrdinalEncoder(),
        DecisionTreeClassifier(
            max_depth=d,
            random_state=42
        )
    )

    model_dt.fit(X_train, y_train)

    training_acc.append(accuracy_score(y_train, model_dt.predict(X_train)))
    validation_acc.append(accuracy_score(y_val, model_dt.predict(X_val)))
```

```
    training_acc.append(accuracy_score(y_train, model_dt.predict(X_train)))
    validation_acc.append(accuracy_score(y_val, model_dt.predict(X_val)))

: submission = pd.Series(validation_acc, index=depth_hyperparams)

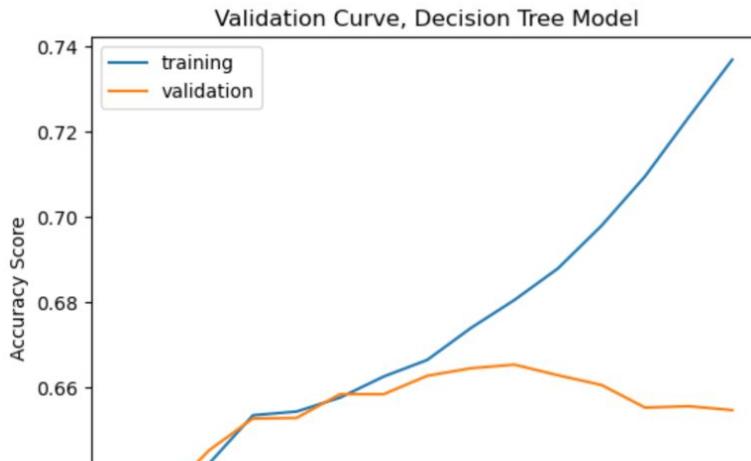
wqet_grader.grade("Project 4 Assessment", "Task 4.5.14", submission)
```

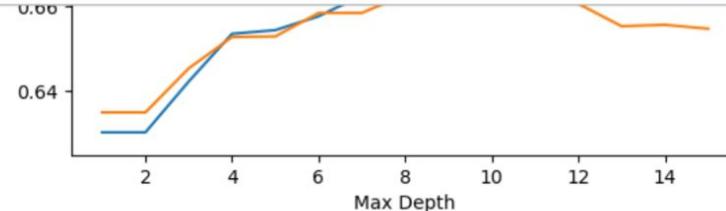
Way to go!

Score: 1

Task 4.5.15: Using the values in `training_acc` and `validation_acc`, plot the validation curve for `model_dt`. Label your x-axis "Max Depth" and your y-axis "Accuracy Score". Use the title "Validation Curve, Decision Tree Model", and include a legend.

```
: plt.plot(depth_hyperparams, training_acc, label="training")
plt.plot(depth_hyperparams, validation_acc, label="validation")
plt.xlabel("Max Depth")
plt.ylabel("Accuracy Score")
plt.title("Validation Curve, Decision Tree Model")
plt.legend();
# Don't delete the code below 🙏
plt.savefig("images/4-5-15.png", dpi=150)
```





```
with open("images/4-5-15.png", "rb") as file:  
    wqet_grader.grade("Project 4 Assessment", "Task 4.5.15", file)
```

Yes! Your hard work is paying off.

Score: 1

Task 4.5.16: Build and train a new decision tree model `final_model_dt`, using the value for `max_depth` that yielded the best validation accuracy score in your plot above.

```
final_model_dt = make_pipeline(OrdinalEncoder(), DecisionTreeClassifier(max_depth=10, random_state=42))
final_model_dt.fit(X_train, y_train)
```

```
Pipeline(steps=[('ordinalencoder',
                 OrdinalEncoder(cols=['land_surface_condition',
                                      'foundation_type', 'roof_type',
                                      'ground_floor_type', 'other_floor_type',
                                      'position', 'plan_configuration',
                                      'superstructure'],
                               mapping=[{'col': 'land_surface_condition',
                                         'data_type': dtype('O'),
                                         'mapping': Flat
                                         }], 1
                                         )]
```

```
Moderate slope      2  
Steep slope        3  
NaN                -2  
dtype: int64
```

```
          {'col': 'foundation_type',
           'dat...
Building with Central Courtyard      9
H-shape                            10
NaN                                -2
dtype: int64},
          {'col': 'superstructure',
```

```
dtype: int64},  
        {'col': 'superstructure',  
         'data_type': dtype('O'),  
         'mapping': Stone, mud mortar      1  
Adobe/mud          2  
Brick, cement mortar 3  
RC, engineered     4  
Brick, mud mortar   5  
Stone, cement mortar 6  
RC, non-engineered 7  
Timber             8  
Other               9  
Bamboo              10  
Stone                11  
NaN                 -2  
dtype: int64]})),  
        ('decisiontreeclassifier',  
         DecisionTreeClassifier(max_depth=10, random_state=42))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
|: wqet_grader.grade("Project 4 Assessment", "Task 4.5.16", final_model_dt)
```

Way to go!

Score: 1

Evaluate

Task 4.5.17: How does your model perform on the test set? First, read the CSV file `"data/kavrepalanchok-test-features.csv"` into the DataFrame `X_test`. Next, use `final_model_dt` to generate a list of test predictions `y_test_pred`. Finally, submit your test predictions to the grader to see how your model performs.

Tip: Make sure the order of the columns in `X_test` is the same as in your `X_train`. Otherwise, it could hurt your model's performance.

```
|: X_test = pd.read_csv("data/kavrepalanchok-test-features.csv", index_col="b_id")  
y_test_pred = final_model_dt.predict(X_test)  
y_test_pred[:5]
```

```
|: array([1, 1, 1, 1, 0])
```

```
y_test_pred[5]: array([1, 1, 1, 1, 0])  
7]: submission = pd.Series(y_test_pred)  
wqet_grader.grade("Project 4 Assessment", "Task 4.5.17", submission)
```

Your model's accuracy score is `0.666`. Yes! Keep on rockin'. ✅ That's right.

Score: 1

Communicate Results

Task 4.5.18: What are the most important features for `final_model_dt`? Create a Series Gini `feat_imp`, where the index labels are the feature names for your dataset and the values are the feature importances for your model. Be sure that the Series is sorted from smallest to largest feature importance.

```
3]: feat_imp = pd.Series(  
    index = X_train.columns,  
    data = final_model_dt.named_steps["decisiontreeclassifier"].feature_importances_  
).sort_values(ascending=True)  
feat_imp.head()
```

```
3]: plan_configuration      0.004189  
land_surface_condition    0.008599  
foundation_type           0.009967  
position                  0.011795  
ground_floor_type         0.013521  
dtype: float64
```

```
1]: wqet_grader.grade("Project 4 Assessment", "Task 4.5.18", feat_imp)
```

Very impressive.

Score: 1

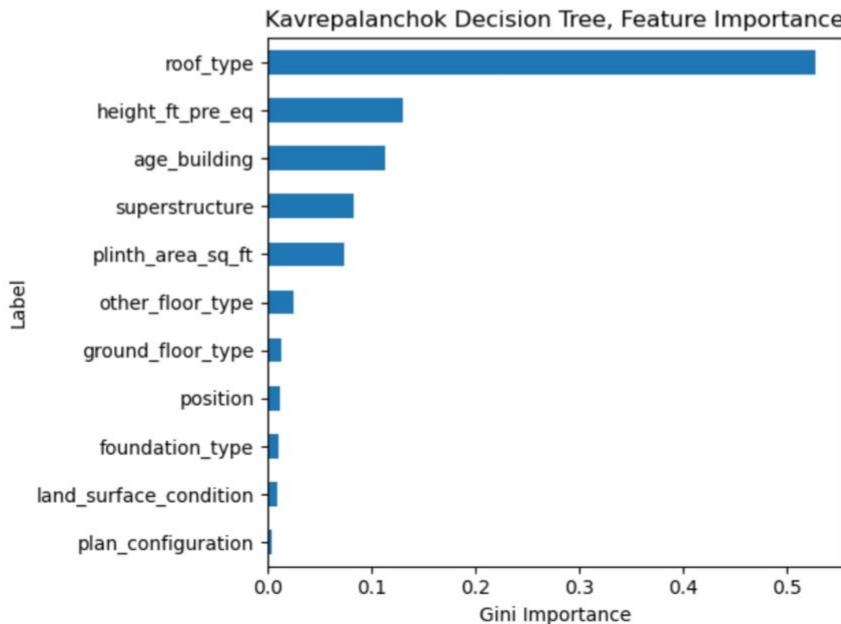
Task 4.5.19: Create a horizontal bar chart of `feat_imp`. Label your x-axis "Gini Importance" and your y-axis "Feature". Use the title "Kavrepalanchok Decision Tree, Feature Importance".

Do you see any relationship between this plot and the exploratory data analysis you did regarding roof type?

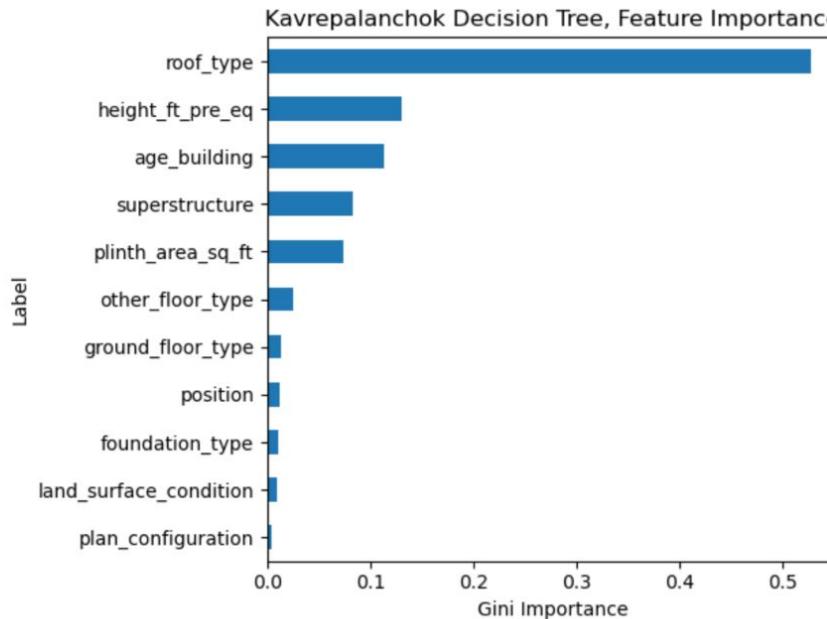
Task 4.5.19: Create a horizontal bar chart of `feat_imp`. Label your x-axis "Gini Importance" and your y-axis "Feature". Use the title "Kavrepalanchok Decision Tree, Feature Importance".

Do you see any relationship between this plot and the exploratory data analysis you did regarding roof type?

```
: # Create horizontal bar chart of feature importances
feat_imp.plot(
    kind="barh",
    xlabel="Gini Importance",
    ylabel="Label",
    title="Kavrepalanchok Decision Tree, Feature Importance"
)
# Don't delete the code below 👉
plt.tight_layout()
plt.savefig("images/4-5-19.png", dpi=150)
```



```
plt.savefig("images/4-5-19.png", dpi=150)
```



```
]: with open("images/4-5-19.png", "rb") as file:  
    wget_grader.grade("Project 4 Assessment", "Task 4.5.19", file)
```



Score: 1

Congratulations! You made it to the end of Project 4. 🎉🎉🎉