045-assignment.2022-06-07T10-46-24-166Z

June 7, 2022

4.5. Earthquake Damage in Kavrepalanchok

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

[50]:
```
import warnings

import wqet_grader

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

```
<IPython.core.display.HTML object>
```

[51]:
```
# Import libraries here
import sqlite3
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from category_encoders import OneHotEncoder
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.linear_model import LogisticRegression
from category_encoders import OrdinalEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
```

1 Prepare Data

1.1 Connect

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

[52]:
```
%load_ext sql
%sql sqlite:////home/jovyan/nepal.sqlite
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

[52]:
```
'Connected: @/home/jovyan/nepal.sqlite'
```

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

[53]:
```
%%sql
SELECT distinct(district_id)
FROM id_map
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[53]:
```
[(1,), (2,), (3,), (4,)]
```

[54]:
```
result = _.DataFrame().squeeze()  # noqa F821

wqet_grader.grade("Project 4 Assessment", "Task 4.5.1", result)
```

```
<IPython.core.display.HTML object>
```

[55]:
```
%%sql
SELECT *
FROM building_damage
LIMIT 5
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[55]:
```
[(1, 'Moderate-Heavy', 'Insignificant/light', 'None', None, 'Moderate-
Heavy-(<1/3)', 'Insignificant/light-(<1/3)', 'Severe-Extreme-(<1/3)', None,
'Insignificant/light-(>2/3)', 'Severe-Extreme-(>2/3)', None, None, 'Severe-
Extreme-(<1/3)', None, None, 'Severe-Extreme-(>2/3)', None, None, None,
'Moderate-Heavy-(>2/3)', None, 'Severe-Extreme-(>2/3)', None, None, 'None',
None, None, 'None', None, None, None, None, None, None, None, None, None,
None, 'None', None, None, 'None', None, None, 'None', None, None, 'Both', 'Grade
3', 'Major repair', 0.0, None, 1.0, 1.0, 1.0, 1.0, None, 1.0, 0.0, 0.0, None,
None, None, 0.0, 0.0, 0.0, 0.0, 0, 0, 0, 0, 0, 0, 0),
 (2, 'Severe-Extreme', 'Severe-Extreme', 'Insignificant/light', 'Severe-
Extreme-(>2/3)', None, None, 'Severe-Extreme-(>2/3)', None, None, 'Severe-
Extreme-(>2/3)', None, None, 'Severe-Extreme-(>2/3)', None, None, 'Severe-
Extreme-(>2/3)', None, None, 'Severe-Extreme-(>2/3)', None, None, 'Severe-
Extreme-(>2/3)', None, None, 'None', None, None, 'Severe-Extreme-(>2/3)', None,
None, None, None, None, None, None, None, None, None, None, 'None', None, None,
'None', None, None, 'None', None, None, 'Exterior', 'Grade 5', 'Reconstruction',
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, None, None, None, 0.0, 0.0,
0.0, 0.0, 0, 0, 0, 0, 0, 0, 0),
 (3, 'Moderate-Heavy', 'Moderate-Heavy', 'Moderate-Heavy', None, 'Moderate-
Heavy-(>2/3)', None, None, 'Moderate-Heavy-(>2/3)', None, None, 'Moderate-
```

```
Heavy-(>2/3)', None, None, 'Moderate-Heavy-(>2/3)', None, None, 'Moderate-
Heavy-(>2/3)', None, None, 'Moderate-Heavy-(>2/3)', None, None, 'Moderate-
Heavy-(>2/3)', None, None, 'Moderate-Heavy-(>2/3)', None, None, None,
'Insignificant/light-(1/3-2/3)', None, None, None, None, None, None, None, None,
None, None, None, None, 'None', None, None, 'None', None, None, 'Both', 'Grade
2', 'Minor repair', 1.0, None, None, None, None, None, None, None, None, None,
None, None, None, 1.0, 0.0, 0.0, 0.0, 0, 0, 0, 0, 0, 0, 0),
 (4, 'Moderate-Heavy', 'Moderate-Heavy', 'Moderate-Heavy', None, 'Moderate-
Heavy-(>2/3)', None, None, 'Moderate-Heavy-(>2/3)', None, None, 'Moderate-
Heavy-(>2/3)', None, None, 'Moderate-Heavy-(>2/3)', None, None, 'Moderate-
Heavy-(>2/3)', None, None, 'Moderate-Heavy-(>2/3)', None, None, 'Moderate-
Heavy-(>2/3)', None, None, None, 'Insignificant/light-(<1/3)', None, None,
'Insignificant/light-(1/3-2/3)', None, None, None, None, None, None, None, None,
None, None, 'Moderate-Heavy-(>2/3)', None, 'None', None, None, 'None', None,
None, 'Both', 'Grade 2', 'Minor repair', 1.0, None, None, None, None, None,
None, None, None, None, None, None, None, None, 0.0, 0.0, 0.0, 0, 0, 0, 0, 0, 0,
0),
 (5, 'Insignificant/light', 'None', 'None', None, None,
'Insignificant/light-(<1/3)', None, None, 'Insignificant/light-(<1/3)', None,
None, 'Insignificant/light-(<1/3)', None, None, 'Insignificant/light-(<1/3)',
None, None, 'Insignificant/light-(<1/3)', None, None,
'Insignificant/light-(<1/3)', None, None, 'Insignificant/light-(<1/3)', 'None',
None, None, 'None', None, None, None, None, None, None, None, None, None, None,
None, 'None', None, None, 'None', None, None, 'None', None, None, 'Exterior',
'Grade 1', 'Minor repair', 1.0, None, None, None, None, None, None, None, 0.0,
0.0, None, None, None, 0.0, 0.0, 0.0, 0.0, 0, 0, 0, 0, 0, 0, 0)]
```

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`.

[56]:
```
%%sql
SELECT count(*)
FROM id_map
WHERE district_id = 1
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[56]:
```
[(36112,)]
```

[58]:
```
result = [_.DataFrame().astype(float).squeeze()]  # noqa F821
wqet_grader.grade("Project 4 Assessment", "Task 4.5.2", result)
```

```
<IPython.core.display.HTML object>
```

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

[59]:
```
%%sql
SELECT count(*)
FROM id_map
WHERE district_id = 3
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[59]:
```
[(82684,)]
```

[60]:
```
result = [_.DataFrame().astype(float).squeeze()]  # noqa F821
wqet_grader.grade("Project 4 Assessment", "Task 4.5.3", result)
```

```
<IPython.core.display.HTML object>
```

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. 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.

[61]:
```
%%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
```

```
 * sqlite:////home/jovyan/nepal.sqlite
Done.
```

[61]:
```
[(87473, 87473, 2, 1, 15, 382, 18, 7, 'Flat', 'Mud mortar-Stone/Brick',
'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached',
'Rectangular', 'Damaged-Used in risk', 'Stone, mud mortar', 'Grade 4'),
 (87479, 87479, 1, 0, 12, 328, 7, 0, 'Flat', 'Mud mortar-Stone/Brick',
'Bamboo/Timber-Light roof', 'Mud', 'Not applicable', 'Not attached',
'Rectangular', 'Damaged-Rubble clear', 'Stone, mud mortar', 'Grade 5'),
 (87482, 87482, 2, 1, 23, 427, 20, 7, 'Flat', 'Mud mortar-Stone/Brick',
'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached',
'Rectangular', 'Damaged-Not used', 'Stone, mud mortar', 'Grade 4'),
 (87491, 87491, 2, 1, 12, 427, 14, 7, 'Flat', 'Mud mortar-Stone/Brick',
'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached',
'Rectangular', 'Damaged-Not used', 'Stone, mud mortar', 'Grade 4'),
 (87496, 87496, 2, 0, 32, 360, 18, 0, 'Flat', 'Mud mortar-Stone/Brick',
'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached',
'Rectangular', 'Damaged-Rubble clear', 'Stone, mud mortar', 'Grade 5')]
```

[62]:
```
result = _.DataFrame().set_index("b_id")  # noqa F821

wqet_grader.grade("Project 4 Assessment", "Task 4.5.4", result)
```

```
<IPython.core.display.HTML object>
```

1.2 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.

[63]:
```
# 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 old target
    drop_cols.append("damage_grade")

    #Drop Multicolinearity Column
    drop_cols.append("count_floors_pre_eq")

    # Drop High Cardinality Categorical  Column
    drop_cols.append("building_id")
```

```
    # drop cols
    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.

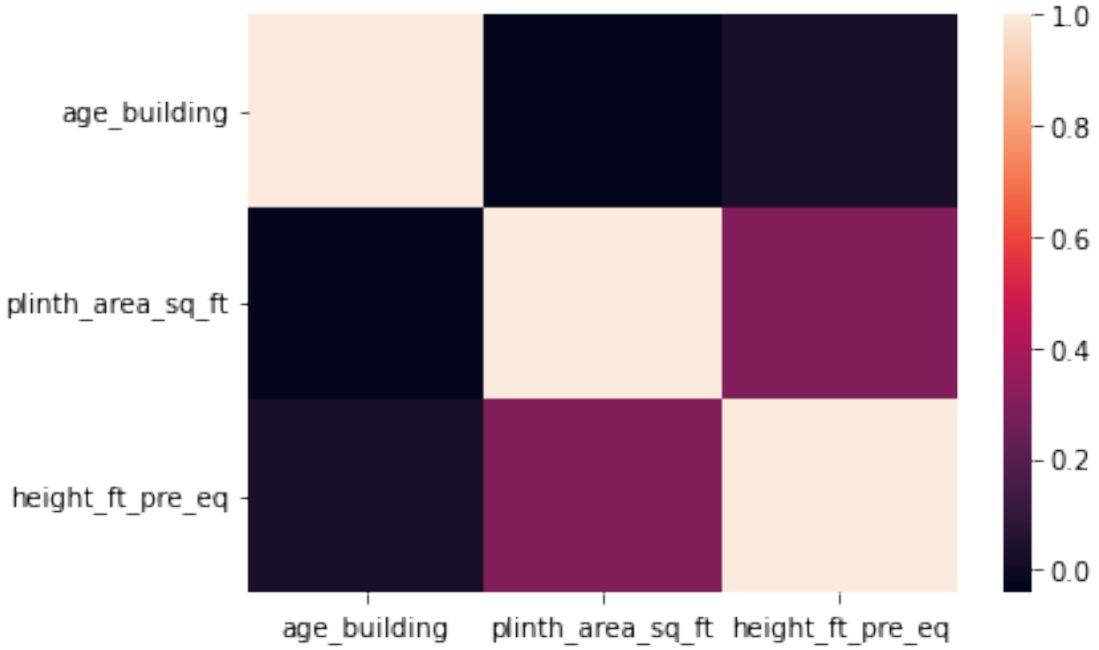
[64]:
```
df = wrangle("/home/jovyan/nepal.sqlite")
df.head()
```

[64]:

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	position	plan_configuration	superstructure	severe_damage
87473	15	382	18	Flat	Mud mortar-Stone/Brick	Bamboo/Timber-Light roof	Mud	TImber/Bamboo-Mud	Not attached	Rectangular	Stone, mud mortar	1
87479	12	328	7	Flat	Mud mortar-Stone/Brick	Bamboo/Timber-Light roof	Mud	Not applicable	Not attached	Rectangular	Stone, mud mortar	1
87482	23	427	20	Flat	Mud mortar-Stone/Brick	Bamboo/Timber-Light roof	Mud	TImber/Bamboo-Mud	Not attached	Rectangular	Stone, mud mortar	1
87491	12	427	14	Flat	Mud mortar-Stone/Brick	Bamboo/Timber-Light roof	Mud	TImber/Bamboo-Mud	Not attached	Rectangular	Stone, mud mortar	1
87496	32	360	18	Flat	Mud mortar-Stone/Brick	Bamboo/Timber-Light roof	Mud	TImber/Bamboo-Mud	Not attached	Rectangular	Stone, mud mortar	1

[65]:
```
# Create correlation matrix
correlation = df.select_dtypes("number").drop(columns="severe_damage").corr()
correlation
# Plot heatmap of `correlation`
sns.heatmap(correlation)
```

[65]: <AxesSubplot:>



[ ]:
```
#df["severe_damage"].corr(df["height_ft_pre_eq"])
#df["severe_damage"].corr(df["count_floors_pre_eq"])
```

[66]:
```
wqet_grader.grade(
    "Project 4 Assessment", "Task 4.5.5", wrange("/home/jovyan/nepal.sqlite")
)
```

```
<IPython.core.display.HTML object>
```

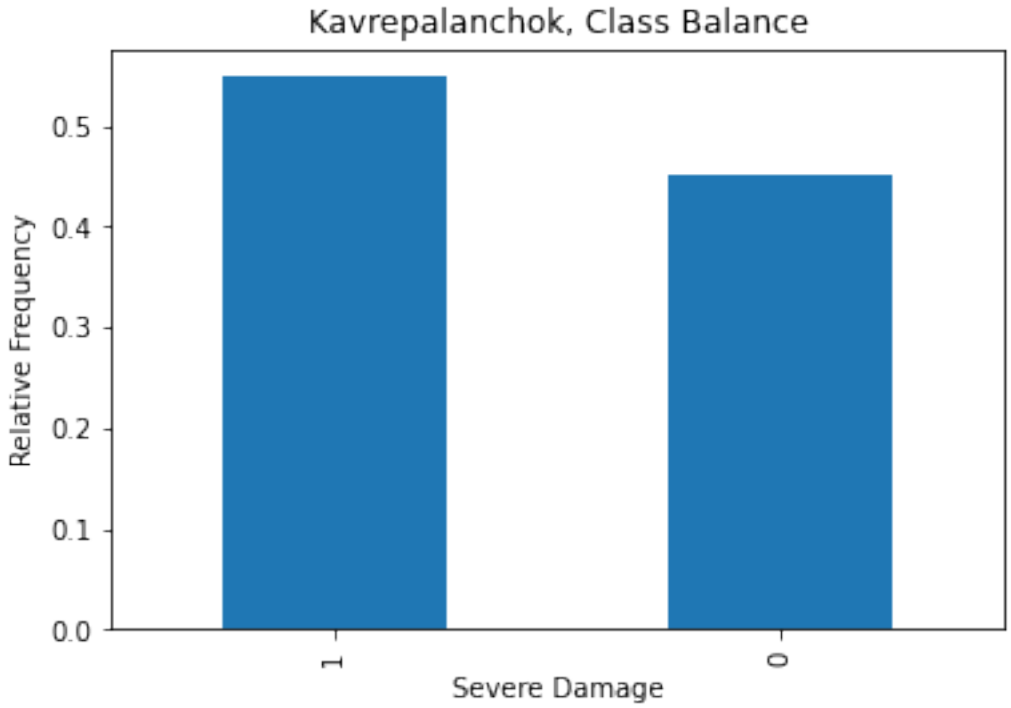
1.3 Explore

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"`.

[67]:
```
# 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
plt.savefig("images/4-5-6.png", dpi=150)
```



[68]:
```
print(df["severe_damage"].value_counts())
```

```
1    41994
0    34539
Name: severe_damage, dtype: int64
```

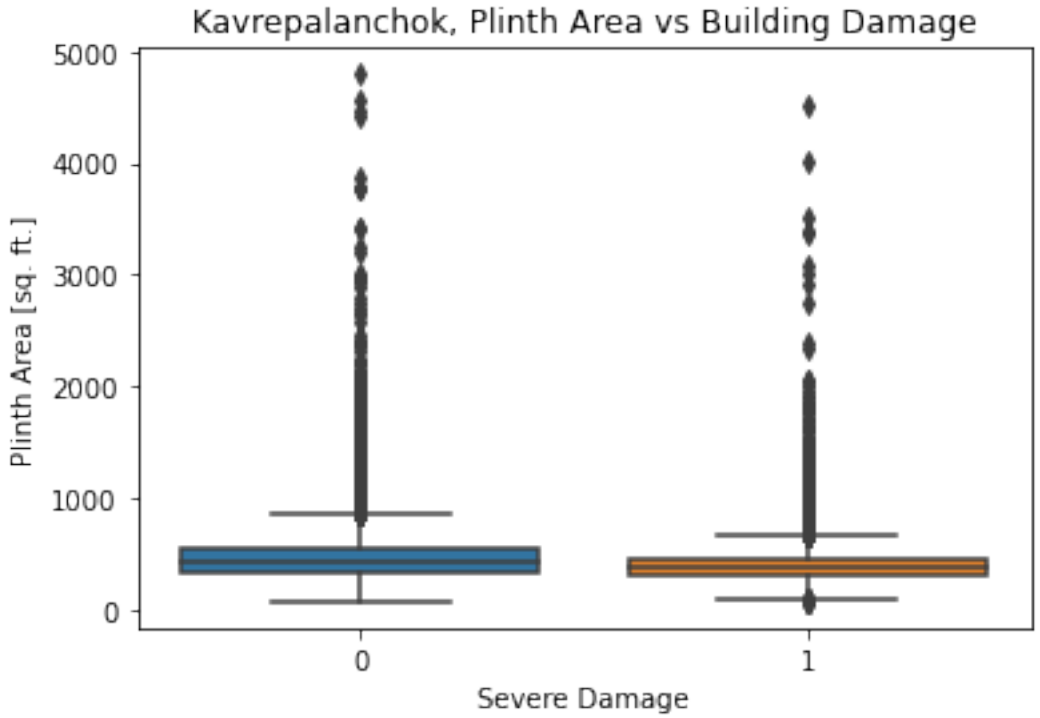
[69]:
```
with open("images/4-5-6.png", "rb") as file:
    wqet_grader.grade("Project 4 Assessment", "Task 4.5.6", file)
```

```
<IPython.core.display.HTML object>
```

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"`.

[70]:
```
# Create boxplot
sns.boxplot(x="severe_damage", y="plinth_area_sq_ft", data=df)
# Label axes
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)
```



[71]:
```
with open("images/4-5-7.png", "rb") as file:
    wqet_grader.grade("Project 4 Assessment", "Task 4.5.7", file)
```

```
<IPython.core.display.HTML object>
```

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.

[72]:
```
roof_pivot = pd.pivot_table(
    df, index="roof_type",values="severe_damage",aggfunc=np.mean
).sort_values(by="severe_damage")
roof_pivot
```

[72]:
roof_type	severe_damage
RCC/RB/RBC	0.040715
Bamboo/Timber-Heavy roof	0.569477
Bamboo/Timber-Light roof	0.604842

[73]:
```
wqet_grader.grade("Project 4 Assessment", "Task 4.5.8", roof_pivot)
```

```
<IPython.core.display.HTML object>
```

1.4 Split

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

[74]:
```
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,)
```

[75]:
```
wqet_grader.grade("Project 4 Assessment", "Task 4.5.9a", X)
```

```
<IPython.core.display.HTML object>
```

[76]:
```
wqet_grader.grade("Project 4 Assessment", "Task 4.5.9b", y)
```

```
<IPython.core.display.HTML object>
```

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.

[77]:
```
# X_train, X_test, y_train, y_test = train_test_split(
#     X,y, test_size=0.2, random_state=42
# )


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,)
```

[78]:
```
wqet_grader.grade("Project 4 Assessment", "Task 4.5.10", [X_train.shape ==␣
↪(61226, 11)])
```

```
<IPython.core.display.HTML object>
```

2 Build Model

2.1 Baseline

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

[79]:
```
acc_baseline = y_train.value_counts(normalize = True).max()
print("Baseline Accuracy:", round(acc_baseline, 2))
```

```
Baseline Accuracy: 0.55
```

[80]:
```
wqet_grader.grade("Project 4 Assessment", "Task 4.5.11", [acc_baseline])
```

```
<IPython.core.display.HTML object>
```

2.2 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.

[81]:
```
model_lr = make_pipeline(
    OneHotEncoder(use_cat_names=True),
    LogisticRegression(max_iter=3000)
)
model_lr.fit(X_train,y_train)
```

[81]:
```
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=3000))])
```

[82]:
```
wqet_grader.grade("Project 4 Assessment", "Task 4.5.12", model_lr)
```

```
<IPython.core.display.HTML object>
```

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

[83]:
```
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: 0.6515042628948486
Logistic Regression, Validation Accuracy Score: 0.6536878552296335
```

[84]:
```
submission = [lr_train_acc, lr_val_acc]
wqet_grader.grade("Project 4 Assessment", "Task 4.5.13", submission)
```

```
<IPython.core.display.HTML object>
```

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.

[85]:
```
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)
     # Calculate training accuracy score and append to `training_acc`
    training_acc.append(model_dt.score(X_train,y_train))
    # Calculate validation accuracy score and append to `training_acc`
    validation_acc.append(model_dt.score(X_val,y_val))

print("Training Accuracy Scores:", training_acc[:3])
print("Validation Accuracy Scores:", validation_acc[:3])
```

```
Training Accuracy Scores: [0.6303041191650606, 0.6303041191650606,
0.642292490118577]
Validation Accuracy Scores: [0.6350035931273273, 0.6350035931273273,
0.6453909975828053]
```

[86]:
```
submission = pd.Series(validation_acc, index=depth_hyperparams)

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

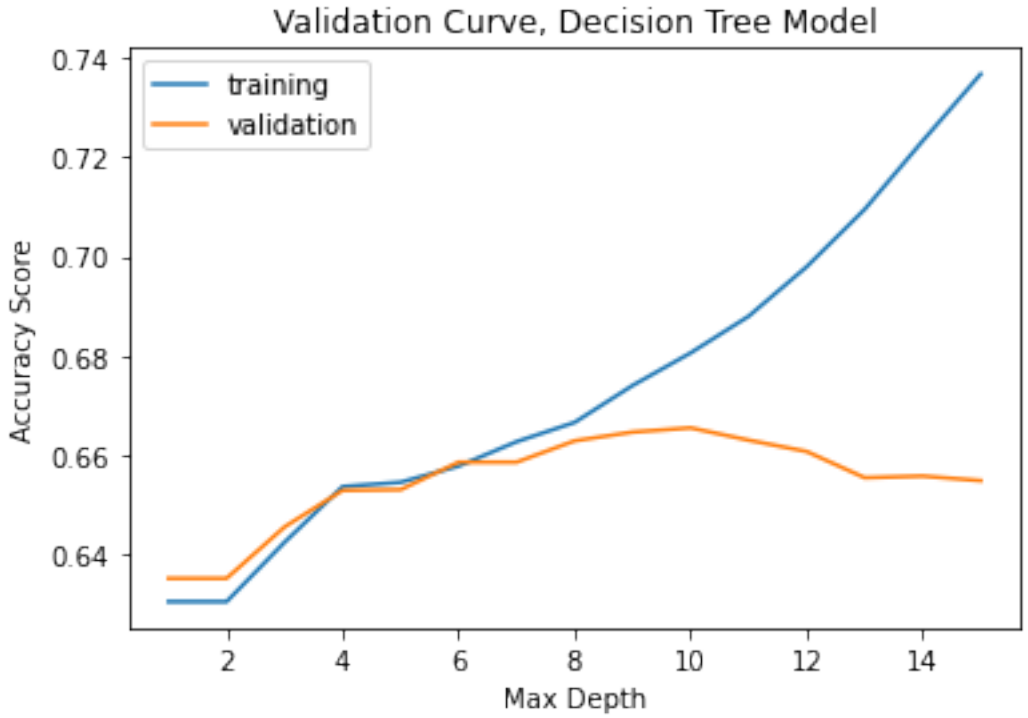
```
<IPython.core.display.HTML object>
```

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.

[87]:
```
# Plot `depth_hyperparams`, `training_acc`
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)
```



[88]:
```
with open("images/4-5-15.png", "rb") as file:
    wqet_grader.grade("Project 4 Assessment", "Task 4.5.15", file)
```

```
<IPython.core.display.HTML object>
```

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.

[97]:
```
final_model_dt = make_pipeline(
    OrdinalEncoder(),
    DecisionTreeClassifier(max_depth=10, random_state=42)
)
final_model_dt.fit(X_val,y_val)
```

[97]:
```
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
Steep slope       2
Moderate slope    3
NaN              -2
dtype: int64},
                                         {'col': 'foundation_type',
                                          'dat…
T-shape            5
H-shape            6
U-shape            7
Others             8
E-shape            9
NaN               -2
dtype: int64},
                                         {'col': 'superstructure',
                                          'data_type': dtype('O'),
                                          'mapping': Stone, mud mortar        1
Adobe/mud                2
RC, non-engineered       3
Brick, cement mortar     4
Brick, mud mortar        5
RC, engineered           6
Bamboo                   7
Timber                   8
Other                    9
Stone                   10
Stone, cement mortar    11
NaN                     -2
dtype: int64}])),
                ('decisiontreeclassifier',
                 DecisionTreeClassifier(max_depth=10, random_state=42))])
```

[98]:
```
wqet_grader.grade("Project 4 Assessment", "Task 4.5.16", final_model_dt)
```

```
<IPython.core.display.HTML object>
```

2.3 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.

[93]:
```
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]
```

[93]:
```
array([1, 1, 1, 1, 0])
```

[99]:
```
submission = pd.Series(y_test_pred)
wqet_grader.grade("Project 4 Assessment", "Task 4.5.17", submission)
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
Input In [99], in <cell line: 2>()
      1 submission = pd.Series(y_test_pred)
----> 2 wqet_grader.grade("Project 4 Assessment", "Task 4.5.17", submission)

File /opt/conda/lib/python3.9/site-packages/wqet_grader/__init__.py:180, in␣
 ↪grade(assessment_id, question_id, submission)
    175 def grade(assessment_id, question_id, submission):
    176   submission_object = {
    177     'type': 'simple',
    178     'argument': [submission]
    179   }
--> 180   return␣
 ↪show_score(grade_submission(assessment_id, question_id, submission_object))

File /opt/conda/lib/python3.9/site-packages/wqet_grader/transport.py:145, in␣
 ↪grade_submission(assessment_id, question_id, submission_object)
    143     raise Exception('Grader raised error: {}'.format(error['message']))
    144   else:
--> 145     raise Exception('Could not grade submission: {}'.
 ↪format(error['message']))
    146 result = envelope['data']['result']
    148 # Used only in testing
```

```
Exception: Could not grade submission: Could not verify access to this␣
 ↪assessment: Received error from WQET submission API: You have already passed␣
 ↪this course!
```

3 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.

[95]:
```
features = X_train.columns
importances = final_model_dt.named_steps["decisiontreeclassifier"].
 ↪feature_importances_

feat_imp =  pd.Series(importances, index=features).sort_values()
feat_imp.head()
```

[95]:
```
plan_configuration         0.004922
land_surface_condition     0.009385
position                   0.014437
roof_type                  0.015071
foundation_type            0.015571
dtype: float64
```

[100]:
```
wqet_grader.grade("Project 4 Assessment", "Task 4.5.18", feat_imp)
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
Input In [100], in <cell line: 1>()
----> 1 wqet_grader.grade("Project 4 Assessment", "Task 4.5.18", feat_imp)

File /opt/conda/lib/python3.9/site-packages/wqet_grader/__init__.py:180, in␣
 ↪grade(assessment_id, question_id, submission)
    175 def grade(assessment_id, question_id, submission):
    176   submission_object = {
    177     'type': 'simple',
    178     'argument': [submission]
    179   }
--> 180   return␣
 ↪show_score(grade_submission(assessment_id, question_id, submission_object))

File /opt/conda/lib/python3.9/site-packages/wqet_grader/transport.py:145, in␣
 ↪grade_submission(assessment_id, question_id, submission_object)
    143     raise Exception('Grader raised error: {}'.format(error['message']))
    144   else:
```

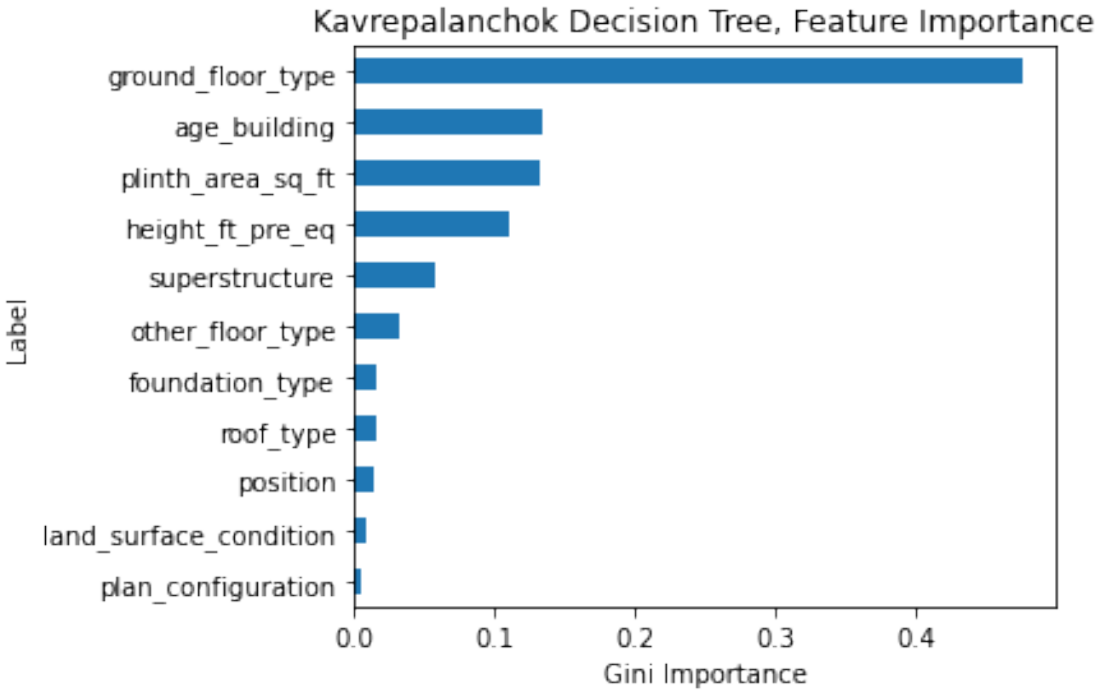
```
--> 145     raise Exception('Could not grade submission: {}'.
 ↪format(error['message']))
    146 result = envelope['data']['result']
    148 # Used only in testing

Exception: Could not grade submission: Could not verify access to this␣
 ↪assessment: Received error from WQET submission API: You have already passed␣
 ↪this course!
```

Task 4.5.19: Create a horizontal bar chart of `feat_imp`. Label your x-axis `"Gini Importance"` and your y-axis `"Label"`. 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?

[96]:
```
# Create horizontal bar chart of feature importances
feat_imp.plot(kind="barh")
plt.xlabel("Gini Importance")
plt.ylabel("Label")
plt.title("Kavrepalanchok Decision Tree, Feature Importance");
# Don't delete the code below
plt.tight_layout()
plt.savefig("images/4-5-19.png", dpi=150)
```



[101]:
```
with open("images/4-5-19.png", "rb") as file:
    wqet_grader.grade("Project 4 Assessment", "Task 4.5.19", file)
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
Input In [101], in <cell line: 1>()
      1 with open("images/4-5-19.png", "rb") as file:
----> 2     wqet_grader.grade("Project 4 Assessment", "Task 4.5.19", file)

File /opt/conda/lib/python3.9/site-packages/wqet_grader/__init__.py:180, in␣
 ↪grade(assessment_id, question_id, submission)
    175 def grade(assessment_id, question_id, submission):
    176   submission_object = {
    177     'type': 'simple',
    178     'argument': [submission]
    179   }
--> 180   return␣
 ↪show_score(grade_submission(assessment_id, question_id, submission_object))

File /opt/conda/lib/python3.9/site-packages/wqet_grader/transport.py:145, in␣
 ↪grade_submission(assessment_id, question_id, submission_object)
    143     raise Exception('Grader raised error: {}'.format(error['message']))
    144   else:
--> 145     raise Exception('Could not grade submission: {}'.
 ↪format(error['message']))
    146 result = envelope['data']['result']
    148 # Used only in testing

Exception: Could not grade submission: Could not verify access to this␣
 ↪assessment: Received error from WQET submission API: You have already passed␣
 ↪this course!
```

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