042-logistic-regression

May 25, 2022

4.2. Predicting Damage with Logistic Regression

[1]:
```
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
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

warnings.simplefilter(action="ignore", category=FutureWarning)
```

[2]:
```
VimeoVideo("665414074", h="d441543f18", width=600)
```

[2]: <IPython.lib.display.VimeoVideo at 0x7fd17c675cd0>

1 Prepare Data

1.1 Import

[3]:
```
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 = 4
    """

    # 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
```

[4]:
```
VimeoVideo("665414541", h="dfe22afdfb", width=600)
```

[4]: <IPython.lib.display.VimeoVideo at 0x7fd09fe517c0>

Task 4.2.1: Complete the `wrangle` function above so that the it returns the results of `query` as a DataFrame. Be sure that the index column is set to `"b_id"`. Also, the path to the SQLite database is `"/home/jovyan/nepal.sqlite"`.

- Read SQL query into a DataFrame using pandas.
- Write a function in Python.

[5]:
```
df = wrangle("/home/jovyan/nepal.sqlite")
df.head()
```

[5]:
```
        age_building  plinth_area_sq_ft  height_ft_pre_eq  \
b_id
164002            20                560                18
164081            21                200                12
```

```
164089                18                 315                 20
164098                45                 290                 13
164103                21                 230                 13

       land_surface_condition         foundation_type  \
b_id
164002                   Flat  Mud mortar-Stone/Brick
164081                   Flat  Mud mortar-Stone/Brick
164089                   Flat  Mud mortar-Stone/Brick
164098                   Flat  Mud mortar-Stone/Brick
164103                   Flat  Mud mortar-Stone/Brick

                       roof_type ground_floor_type   other_floor_type  \
b_id
164002  Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud
164081  Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud
164089  Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud
164098  Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud
164103  Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud

            position plan_configuration     superstructure  severe_damage
b_id
164002  Not attached        Rectangular  Stone, mud mortar              0
164081  Not attached        Rectangular  Stone, mud mortar              0
164089  Not attached        Rectangular  Stone, mud mortar              0
164098  Not attached        Rectangular  Stone, mud mortar              0
164103  Not attached        Rectangular  Stone, mud mortar              0
```

[6]:
```python
# Check your work
assert df.shape[0] == 70836, f"`df` should have 70,836 rows, not {df.shape[0]}."
```

There seem to be several features in `df` with information about the condition of a property after the earthquake.

[7]:
```python
VimeoVideo("665414560", h="ad4bba19ed", width=600)
```

[7]:
```
<IPython.lib.display.VimeoVideo at 0x7fd09b24c730>
```

[8]:
```python
# drop_cols = []
# for col in df.columns:
#     if"post_eq" in col:
#         drop_cols.append(col)

# Let's try with List_comprehension of same things just done before
# drop_cols = [col for col in df.columns if "post_eq" in col]
# drop_cols
```

Task 4.2.2: Add to your wrangle function so that these features are dropped from the DataFrame.

Don't forget to rerun all the cells above.

- Drop a column from a DataFrame using pandas.
- Subset a DataFrame's columns based on column names in pandas.

[9]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 70836 entries, 164002 to 234835
Data columns (total 12 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   age_building           70836 non-null  int64
 1   plinth_area_sq_ft      70836 non-null  int64
 2   height_ft_pre_eq       70836 non-null  int64
 3   land_surface_condition 70836 non-null  object
 4   foundation_type        70836 non-null  object
 5   roof_type              70836 non-null  object
 6   ground_floor_type      70836 non-null  object
 7   other_floor_type       70836 non-null  object
 8   position               70836 non-null  object
 9   plan_configuration     70836 non-null  object
 10  superstructure         70836 non-null  object
 11  severe_damage          70836 non-null  int64
dtypes: int64(4), object(8)
memory usage: 7.0+ MB
None
```

[10]:
```
# Check your work
assert (
    df.filter(regex="post_eq").shape[1] == 0
), "`df` still has leaky features. Try again!"
```

We want to build a **binary classification** model, but our current target `"damage_grade"` has more than two categories.

[11]:
```
VimeoVideo("665414603", h="12b3d2f23e", width=600)
```

[11]: `<IPython.lib.display.VimeoVideo at 0x7fd09f284670>`

[12]:
```
# df["damage_grade"].head()
# df["damage_grade"].value_counts()

# df["damage_grade"] = df["damage_grade"].str[-1].astype(int)
# df["severe_damage"] = (df["damage_grade"] > 3).astype(int).head(10)
```

Task 4.2.3: Add to your wrangle function so that it creates a new target column `"severe_damage"`. For buildings where the `"damage_grade"` is Grade 4 or above, `"severe_damage"` should be `1`. For all other buildings, "severe_damage" should be 0. Don't forget to drop "damage_grade" to avoid leakage, and rerun all the cells above.

- Access a substring in a Series using pandas.
- Drop a column from a DataFrame using pandas.
- Recast a column as a different data type in pandas.

[13]:
```
print(df["severe_damage"].value_counts())
```

```
1    45519
0    25317
Name: severe_damage, dtype: int64
```

[14]:
```
# Check your work
assert (
    "damage_grade" not in df.columns
), "Your DataFrame should not include the `'damage_grade'` column."
assert (
    "severe_damage" in df.columns
), "Your DataFrame is missing the `'severe_damage'` column."
assert (
    df["severe_damage"].value_counts().shape[0] == 2
), f"The `'damage_grade'` column should have only two unique values, not␣
 ↪{df['severe_damage'].value_counts().shape[0]}"
```

1.2 Explore

Since our model will be a type of linear model, we need to make sure there's no issue with multicollinearity in our dataset.

[15]:
```
VimeoVideo("665414636", h="d34256b4e3", width=600)
```

[15]: <IPython.lib.display.VimeoVideo at 0x7fd09f284dc0>

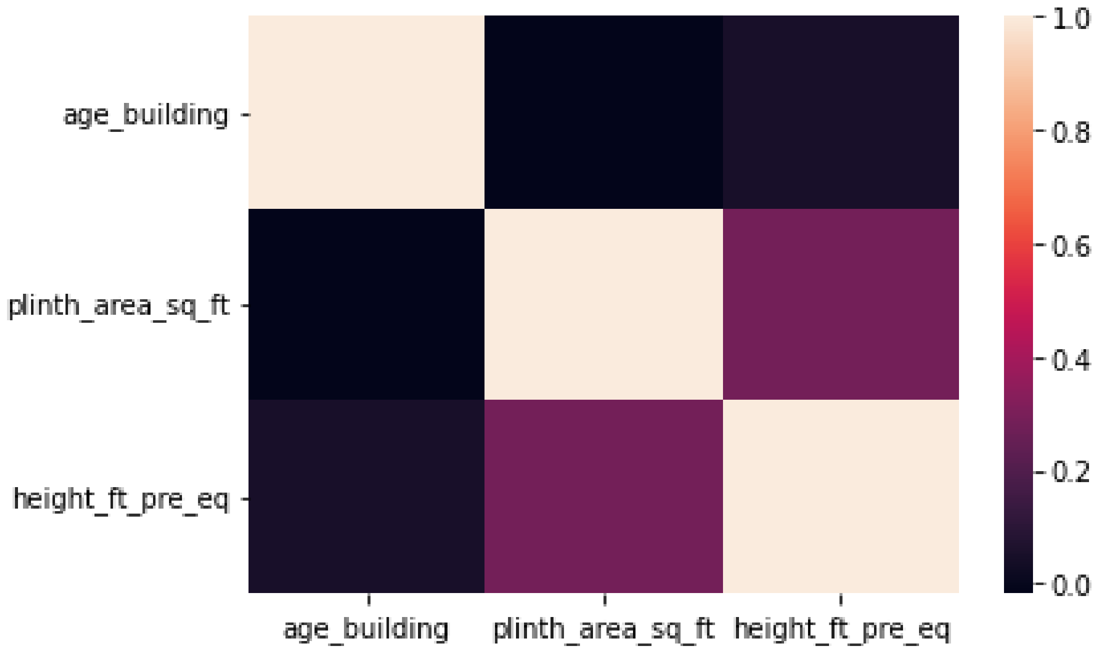
Task 4.2.4: Plot a correlation heatmap of the remaining numerical features in df. Since "severe_damage" will be your target, you don't need to include it in your heatmap.

- What's a correlation coefficient?
- What's a heatmap?
- Create a correlation matrix in pandas.
- Create a heatmap in seaborn.

Do you see any features that you need to drop?

[16]:
```
# Create correlation matrix
correlation = df.select_dtypes("number").drop(columns="severe_damage").corr()
correlation
# Plot heatmap of `correlation`
sns.heatmap(correlation)
```

[16]: <AxesSubplot:>



[17]:
```
#df["severe_damage"].corr(df["height_ft_pre_eq"])
#df["severe_damage"].corr(df["count_floors_pre_eq"])
```

Task 4.2.5: Change `wrangle` function so that it drops the `"count_floors_pre_eq"` column. Don't forget to rerun all the cells above.

- Drop a column from a DataFrame using pandas.

[18]:
```
# Check your work
assert (
    "count_floors_pre_eq" not in df.columns
), "Did you drop the `'count_floors_pre_eq'` column?"
```

Before we build our model, let's see if we can identify any obvious differences between houses that were severely damaged in the earthquake (`"severe_damage"==1`) those that were not (`"severe_damage"==0`). Let's start with a numerical feature.

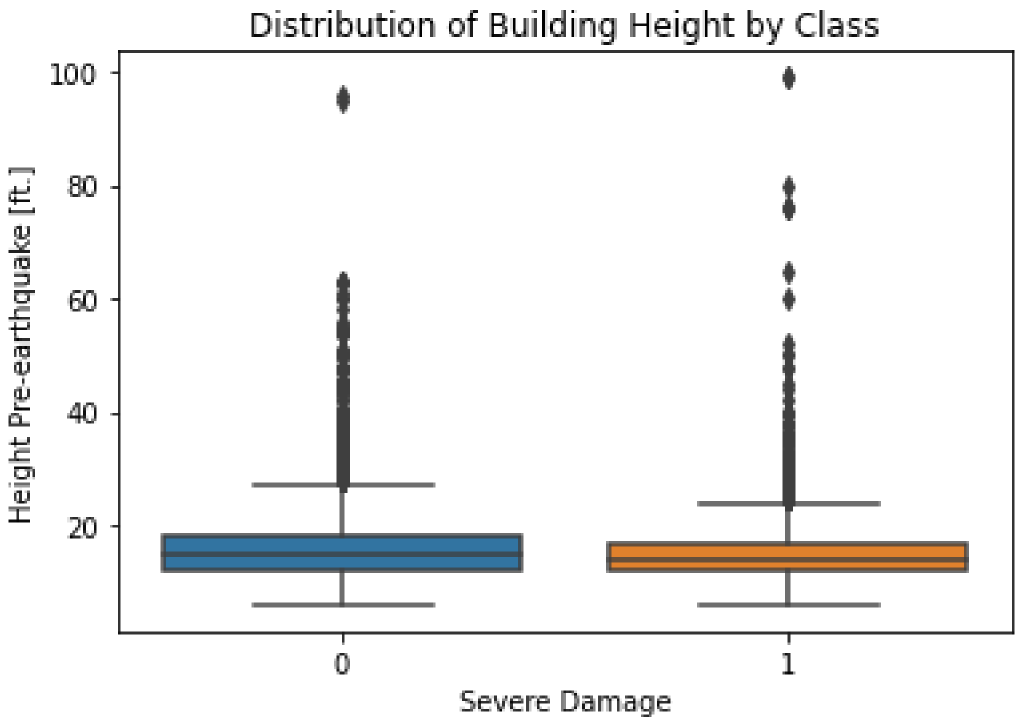
[19]:
```
VimeoVideo("665414667", h="f39c2c21bc", width=600)
```

[19]: <IPython.lib.display.VimeoVideo at 0x7fd09f284970>

Task 4.2.6: Use seaborn to create a boxplot that shows the distributions of the `"height_ft_pre_eq"` column for both groups in the `"severe_damage"` column. Remember to label your axes.

- What's a boxplot?
- Create a boxplot using Matplotlib.

[20]:
```
# Create boxplot
sns.boxplot(x="severe_damage", y="height_ft_pre_eq", data=df)
# Label axes
plt.xlabel("Severe Damage")
plt.ylabel("Height Pre-earthquake [ft.]")
plt.title("Distribution of Building Height by Class");
```



Before we move on to the many categorical features in this dataset, it's a good idea to see the balance between our two classes. What percentage were severely damaged, what percentage were not?

[21]:
```
VimeoVideo("665414684", h="81295d5bdb", width=600)
```

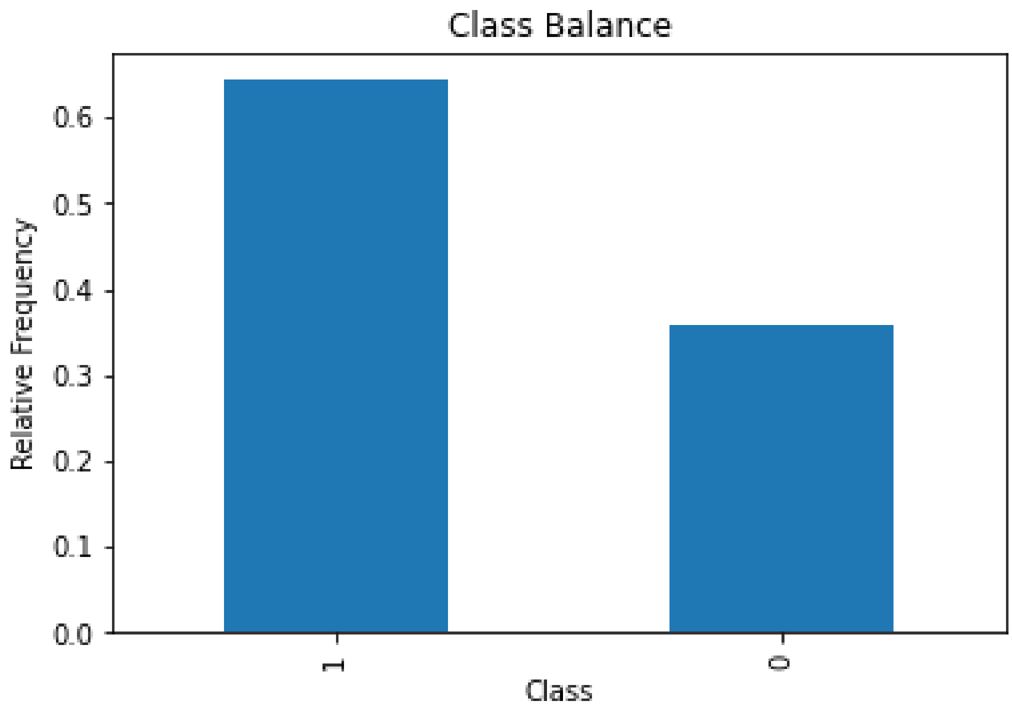
[21]: <IPython.lib.display.VimeoVideo at 0x7fd09e69d8b0>

Task 4.2.7: Create a bar chart of the value counts for the `"severe_damage"` column. You want to calculate the relative frequencies of the classes, not the raw count, so be sure to set the `normalize` argument to `True`.

- What's a bar chart?
- What's a majority class?
- What's a minority class?

- Aggregate data in a Series using `value_counts` in pandas.
- Create a bar chart using pandas.

[22]:
```
# Plot value counts of `"severe_damage"`
df["severe_damage"].value_counts(normalize = True).plot(
    kind="bar", xlabel="Class", ylabel="Relative Frequency", title="Class␣
↪Balance"
);
```



[23]:
```
VimeoVideo("665414697", h="ee2d4f28c6", width=600)
```

[23]: <IPython.lib.display.VimeoVideo at 0x7fd12b7d77f0>

Task 4.2.8: Create two variables, `majority_class_prop` and `minority_class_prop`, to store the normalized value counts for the two classes in `df["severe_damage"]`.

- Aggregate data in a Series using `value_counts` in pandas.

[24]:
```
majority_class_prop, minority_class_prop = df["severe_damage"].
↪value_counts(normalize = True)
print(majority_class_prop, minority_class_prop)
```

```
0.6425969845841097 0.3574030154158902
```

[25]:
```
# Check your work
assert (
    majority_class_prop < 1
), "`majority_class_prop` should be a floating point number between 0 and 1."
assert (
    minority_class_prop < 1
), "`minority_class_prop` should be a floating point number between 0 and 1."
```

[26]:
```
VimeoVideo("665414718", h="6a1e0c1e53", width=600)
```

[26]: <IPython.lib.display.VimeoVideo at 0x7fd12b7d7c40>

Task 4.2.9: Are buildings with certain foundation types more likely to suffer severe damage? Create a pivot table of `df` where the index is `"foundation_type"` and the values come from the `"severe_damage"` column, aggregated by the mean.

- What's a pivot table?
- Reshape a DataFrame based on column values in pandas.

[27]:
```
# Create pivot table
foundation_pivot = pd.pivot_table(
    df, index="foundation_type",values="severe_damage",aggfunc=np.mean
).sort_values(by="severe_damage")
foundation_pivot
```

[27]:

	severe_damage
foundation_type	
RC	0.026224
Bamboo/Timber	0.324074
Cement-Stone/Brick	0.421908
Mud mortar-Stone/Brick	0.687792
Other	0.818898

[28]:
```
VimeoVideo("665414734", h="46de83f96e", width=600)
```

[28]: <IPython.lib.display.VimeoVideo at 0x7fd12b7d9430>

Task 4.2.10: How do the proportions in `foundation_pivot` compare to the proportions for our majority and minority classes? Plot `foundation_pivot` as horizontal bar chart, adding vertical lines at the values for `majority_class_prop` and `minority_class_prop`.

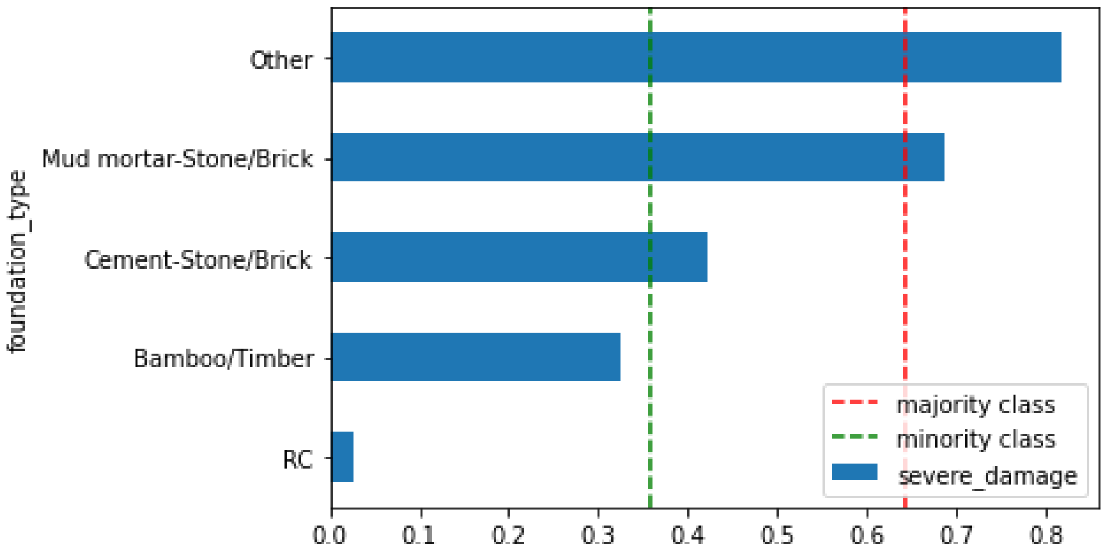
- What's a bar chart?
- Add a vertical or horizontal line across a plot using Matplotlib.
- Create a bar chart using pandas.

[29]:
```
# Plot bar chart of `foundation_pivot`
foundation_pivot.plot(kind="barh",legend=None)
plt.axvline(
    majority_class_prop, linestyle="--",color="red" , label="majority class"
```

```
)

plt.axvline(
    minority_class_prop, linestyle="--",color="green", label="minority class"
)
plt.legend(loc="lower right")
```

[29]: <matplotlib.legend.Legend at 0x7fd09f3ea160>



[30]:
```
VimeoVideo("665414748", h="8549a0f89c", width=600)
```

[30]: <IPython.lib.display.VimeoVideo at 0x7fd09f2d6e20>

Task 4.2.11: Combine the `select_dtypes` and `nunique` methods to see if there are any high- or low-cardinality categorical features in the dataset.

- What are high- and low-cardinality features?
- Determine the unique values in a column using pandas.
- Subset a DataFrame's columns based on the column data types in pandas.

[31]:
```
# Check for high- and low-cardinality categorical features
df.select_dtypes("object").nunique()
```

[31]:
```
land_surface_condition     3
foundation_type            5
roof_type                  3
ground_floor_type          5
other_floor_type           4
position                   4
```

```
plan_configuration        10
superstructure            11
dtype: int64
```

1.3 Split

Task 4.2.12: Create your feature matrix `X` and target vector `y`. Your target is `"severe_damage"`.

- What's a feature matrix?
- What's a target vector?
- Subset a DataFrame by selecting one or more columns in pandas.
- Select a Series from a DataFrame in pandas.

[32]:
```
target = "severe_damage"
X = df.drop(columns=target)
y = df[target]
```

[33]:
```
VimeoVideo("665414769", h="1bfddf07b2", width=600)
```

[33]: <IPython.lib.display.VimeoVideo at 0x7fd09f2d6c40>

Task 4.2.13: Divide your data (`X` and `y`) into training and test sets using a randomized train-test split. Your test set should be 20% of your total data. And don't forget to set a `random_state` for reproducibility.

- Perform a randomized train-test split using scikit-learn.

[34]:
```
X_train, X_test, y_train, y_test = 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_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (56668, 11)
y_train shape: (56668,)
X_test shape: (14168, 11)
y_test shape: (14168,)
```

2 Build Model

2.1 Baseline

[35]:
```
VimeoVideo("665414807", h="c997c58720", width=600)
```

[35]: <IPython.lib.display.VimeoVideo at 0x7fd09f301ac0>

Task 4.2.14: Calculate the baseline accuracy score for your model.

- What's accuracy score?
- Aggregate data in a Series using `value_counts` in pandas.

[36]:
```
acc_baseline = y_train.value_counts(normalize=True).max()
print("Baseline Accuracy:", round(acc_baseline, 2))
```

```
Baseline Accuracy: 0.64
```

2.2 Iterate

[37]:
```
VimeoVideo("665414835", h="1d8673223e", width=600)
```

[37]: `<IPython.lib.display.VimeoVideo at 0x7fd09f3014c0>`

Task 4.2.15: Create a pipeline named `model` that contains a `OneHotEncoder` transformer and a `LogisticRegression` predictor. Be sure you set the `use_cat_names` argument for your transformer to `True`. Then fit it to the training data.

- What's logistic regression?
- What's one-hot encoding?
- Create a pipeline in scikit-learn.
- Fit a model to training data in scikit-learn.

Tip: If you get a ConvergenceWarning when you fit your model to the training data, don't worry. This can sometimes happen with logistic regression models. Try setting the max_iter argument in your predictor to 1000.

[38]:
```
# Build model
model = make_pipeline(
    OneHotEncoder(use_cat_names=True),
    LogisticRegression(max_iter=1000)
)
# Fit model to training data
model.fit(X_train, y_train)
```

```
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[38]:
```
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))])
```

[39]:
```
# Check your work
assert isinstance(
    model, Pipeline
), f"`model` should be a Pipeline, not type {type(model)}."
assert isinstance(
    model[0], OneHotEncoder
), f"The first step in your Pipeline should be a OneHotEncoder, not type␣
 ↪{type(model[0])}."
assert isinstance(
    model[-1], LogisticRegression
), f"The last step in your Pipeline should be LogisticRegression, not type␣
 ↪{type(model[-1])}."
check_is_fitted(model)
```

2.3 Evaluate

[40]:
```
VimeoVideo("665414885", h="f35ff0e23e", width=600)
```

[40]:
```
<IPython.lib.display.VimeoVideo at 0x7fd09f404cd0>
```

Task 4.2.16: Calculate the training and test accuracy scores for your models.

- Calculate the accuracy score for a model in scikit-learn.
- Generate predictions using a trained model in scikit-learn.

[41]:
```
acc_train = accuracy_score(y_train, model.predict(X_train))
acc_test = model.score(X_test, y_test)

print("Training Accuracy:", round(acc_train, 2))
print("Test Accuracy:", round(acc_test, 2))
```

```
Training Accuracy: 0.71
Test Accuracy: 0.72
```

3 Communicate

[42]:
```
VimeoVideo("665414902", h="f9bdbe9e75", width=600)
```

[42]:
```
<IPython.lib.display.VimeoVideo at 0x7fd09f2d6ee0>
```

Task 4.2.17: Instead of using the `predict` method with your model, try `predict_proba` with your training data. How does the `predict_proba` output differ than that of `predict`? What does it represent?

- Generate probability estimates using a trained model in scikit-learn.

[48]:
```
#y_train_pred = model.predict(X_train)
y_train_pred_proba = model.predict_proba(X_train)
print(y_train_pred_proba[:5])
```

```
[[0.96799838 0.03200162]
 [0.48832971 0.51167029]
 [0.34807011 0.65192989]
 [0.39326981 0.60673019]
 [0.33352859 0.66647141]]
```

Task 4.2.18: Extract the feature names and importances from your `model`.

- Access an object in a pipeline in scikit-learn.

[51]:
```
features = model.named_steps["onehotencoder"].get_feature_names()
importances = model.named_steps["logisticregression"].coef_[0]
```

[49]:
```
VimeoVideo("665414916", h="c0540604cd", width=600)
```

[49]:
```
<IPython.lib.display.VimeoVideo at 0x7fd09d0aa460>
```

Task 4.2.19: Create a pandas Series named `odds_ratios`, where the index is `features` and the values are your the exponential of the `importances`. How does `odds_ratios` for this model look different from the other linear models we made in projects 2 and 3?

- Create a Series in pandas.

[53]:
```
odds_ratios = pd.Series(np.exp(importances), index=features).sort_values()
odds_ratios.head()
```

[53]:
```
superstructure_Brick, cement mortar    0.243018
foundation_type_RC                     0.338894
roof_type_RCC/RB/RBC                   0.387966
ground_floor_type_RC                   0.484377
plan_configuration_Multi-projected     0.551307
dtype: float64
```

[54]:
```
VimeoVideo("665414943", h="56eb74d93e", width=600)
```

[54]:
```
<IPython.lib.display.VimeoVideo at 0x7fd09c6f65e0>
```

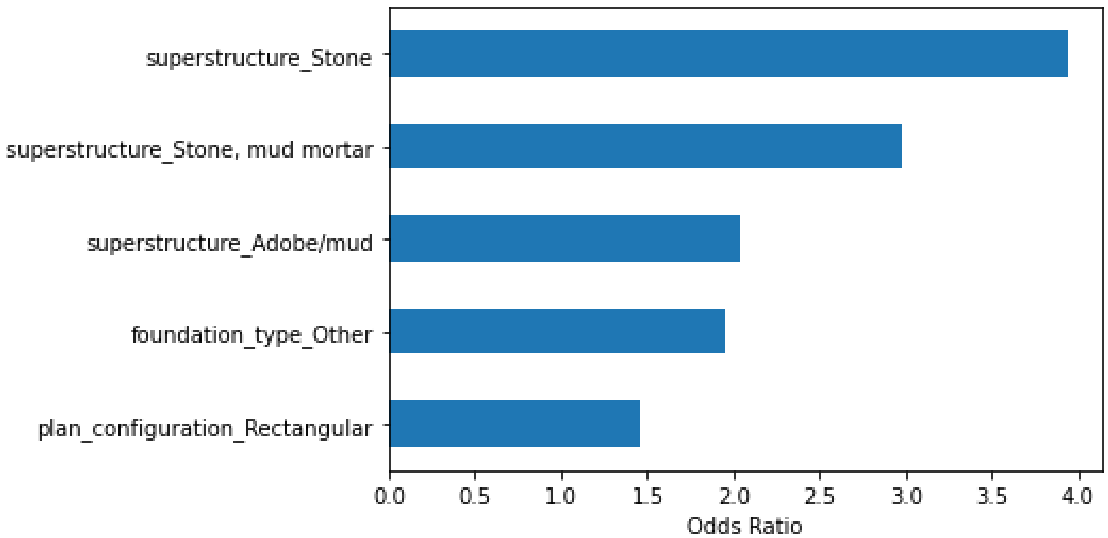
Task 4.2.20: Create a horizontal bar chart with the five largest coefficients from `odds_ratios`. Be sure to label your x-axis `"Odds Ratio"`.

- What's a bar chart?

- Create a bar chart using Matplotlib.

[57]:
```
# Horizontal bar chart, five largest coefficients
odds_ratios.tail().plot(kind="barh")
plt.xlabel("Odds Ratio")
```

[57]: Text(0.5, 0, 'Odds Ratio')



[58]:
```
VimeoVideo("665414964", h="a61b881450", width=600)
```

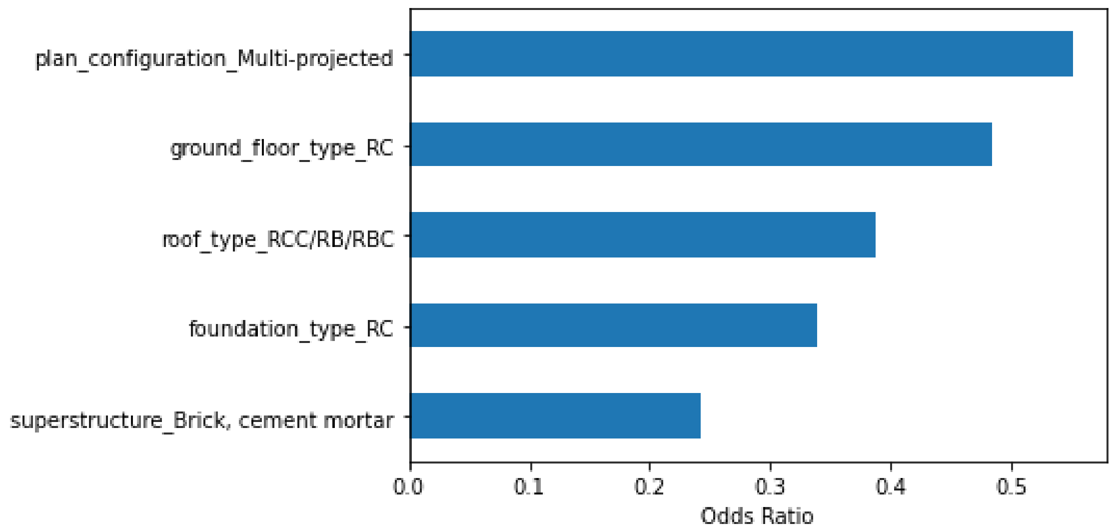
[58]: <IPython.lib.display.VimeoVideo at 0x7fd094c92310>

Task 4.2.21: Create a horizontal bar chart with the five smallest coefficients from `odds_ratios`. Be sure to label your x-axis `"Odds Ratio"`.

- What's a bar chart?
- Create a bar chart using Matplotlib.

[59]:
```
# Horizontal bar chart, five smallest coefficients
odds_ratios.head().plot(kind="barh")
plt.xlabel("Odds Ratio")
```

[59]: Text(0.5, 0, 'Odds Ratio')


plan_configuration_Multi-projected
ground_floor_type_RC
roof_type_RCC/RB/RBC
foundation_type_RC
superstructure_Brick, cement mortar
0.0
0.1
0.2
0.3
0.4
0.5
Odds Ratio