044-demographics.2022-06-07T10-46-24-166Z

June 7, 2022

4.4. Beyond the Model: Data Ethics

[2]:
```
import sqlite3
import warnings

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
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)
```

[3]:
```
VimeoVideo("665414155", h="c8a3e81a05", width=600)
```

[3]:
```
<IPython.lib.display.VimeoVideo at 0x7f052c45eee0>
```

1 Prepare Data

Task 4.4.1: Run the cell below to connect to the `nepal.sqlite` database.

- What's ipython-sql?
- What's a Magics function?

[4]:
```
%load_ext sql
%sql sqlite:////home/jovyan/nepal.sqlite
```

[4]:
```
'Connected: @/home/jovyan/nepal.sqlite'
```

[5]:
```
VimeoVideo("665415362", h="f677c48c46", width=600)
```

[5]:
```
<IPython.lib.display.VimeoVideo at 0x7f04794adeb0>
```

Task 4.4.2: Select all columns from the `household_demographics` table, limiting your results to the first five rows.

- Write a basic query in SQL.
- Inspect a table using a `LIMIT` clause in SQL.

[6]:
```
%%sql
SELECT *
FROM household_demographics
LIMIT 5
```

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

[6]:
```
[(101, 'Male', 31.0, 'Rai', 'Illiterate', 'Rs. 10 thousand', 3.0, 0.0),
 (201, 'Female', 62.0, 'Rai', 'Illiterate', 'Rs. 10 thousand', 6.0, 0.0),
 (301, 'Male', 51.0, 'Gharti/Bhujel', 'Illiterate', 'Rs. 10 thousand', 13.0,
0.0),
 (401, 'Male', 48.0, 'Gharti/Bhujel', 'Illiterate', 'Rs. 10 thousand', 5.0,
0.0),
 (501, 'Male', 70.0, 'Gharti/Bhujel', 'Illiterate', 'Rs. 10 thousand', 8.0,
0.0)]
```

Task 4.4.3: How many observations are in the `household_demographics` table? Use the `count` command to find out.

- Calculate the number of rows in a table using a `count` function in SQL.

[7]:
```
%%sql
SELECT count(*)
FROM household_demographics
```

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

[7]:
```
[(249932,)]
```

[8]:
```
VimeoVideo("665415378", h="aa2b99493e", width=600)
```

[8]:
```
<IPython.lib.display.VimeoVideo at 0x7f04794d5d00>
```

Task 4.4.4: Select all columns from the `id_map` table, limiting your results to the first five rows.

- Inspect a table using a `LIMIT` clause in SQL.

What columns does it have in common with `household_demographics` that we can use to join them?

[9]:
```
%%sql
SELECT *
FROM id_map
```

```
LIMIT 5
```

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

[9]:
```
[(5601, 56, 7, 1),
 (6301, 63, 7, 1),
 (9701, 97, 7, 1),
 (9901, 99, 7, 1),
 (11501, 115, 7, 1)]
```

[10]:
```
VimeoVideo("665415406", h="46a990c8f7", width=600)
```

[10]:
```
<IPython.lib.display.VimeoVideo at 0x7f04794adb80>
```

Task 4.4.5: Create a table with all the columns from `household_demographics`, all the columns from `building_structure`, the **`vdcmun_id`** column from `id_map`, and the **`damage_grade`** column from `building_damage`. Your results should show only rows where the **`district_id`** is 5 and limit your results to the first five rows.

- Create an alias for a column or table using the `AS` command in SQL.
- Determine the unique values in a column using a `DISTINCT` function in SQL.
- Merge two tables using a `JOIN` clause in SQL.
- Inspect a table using a `LIMIT` clause in SQL.
- Subset a table using a `WHERE` clause in SQL.

[11]:
```
%%sql
SELECT h.*,
       s.*,
       i.vdcmun_id,
       d.damage_grade
FROM household_demographics AS h
JOIN id_map AS i ON i.household_id = h.household_id
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
LIMIT 5
```

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

[11]:
```
[(16400201, 'Female', 46.0, 'Chhetree', 'Class 5', 'Rs. 10-20 thousand', 4.0,
1.0, 164002, 3, 3, 20, 560, 18, 18, 'Flat', 'Mud mortar-Stone/Brick',
'Bamboo/Timber-Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached',
'Rectangular', 'Damaged-Repaired and used', 'Stone, mud mortar', 38, 'Grade 2'),
 (16408101, 'Male', 66.0, 'Chhetree', 'Illiterate', 'Rs. 10 thousand', 5.0, 0.0,
164081, 2, 2, 21, 200, 12, 12, 'Flat', 'Mud mortar-Stone/Brick', 'Bamboo/Timber-
Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached', 'Rectangular',
```

```
'Damaged-Used in risk', 'Stone, mud mortar', 38, 'Grade 2'),
 (16408901, 'Male', 54.0, 'Magar', 'Class 4', 'Rs. 10 thousand', 5.0, 1.0,
164089, 3, 3, 18, 315, 20, 20, 'Flat', 'Mud mortar-Stone/Brick', 'Bamboo/Timber-
Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached', 'Rectangular',
'Damaged-Used in risk', 'Stone, mud mortar', 38, 'Grade 2'),
 (16409801, 'Male', 36.0, 'Chhetree', 'Class 5', 'Rs. 10 thousand', 6.0, 1.0,
164098, 2, 2, 45, 290, 13, 13, 'Flat', 'Mud mortar-Stone/Brick', 'Bamboo/Timber-
Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached', 'Rectangular',
'Damaged-Used in risk', 'Stone, mud mortar', 38, 'Grade 3'),
 (16410301, 'Female', 39.0, 'Chhetree', 'Class 4', 'Rs. 10 thousand', 3.0, 0.0,
164103, 2, 2, 21, 230, 13, 13, 'Flat', 'Mud mortar-Stone/Brick', 'Bamboo/Timber-
Light roof', 'Mud', 'TImber/Bamboo-Mud', 'Not attached', 'Rectangular',
'Damaged-Used in risk', 'Stone, mud mortar', 38, 'Grade 3')]
```

1.1 Import

[12]:
```python
def wrangle(db_path):
    # Connect to database
    conn = sqlite3.connect(db_path)

    # Construct query
    query = """
    SELECT h.*,
           s.*,
           i.vdcmun_id,
           d.damage_grade
    FROM household_demographics AS h
    JOIN id_map AS i ON i.household_id = h.household_id
    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="household_id")

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

    # Add high-cardinality / redundant column
    drop_cols.append("building_id")

    # Create binary target column
    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 multicollinearity column
    drop_cols.append("count_floors_pre_eq")

    #Group caste columns
    top_10 = df["caste_household"].value_counts().head(10).index
    df["caste_household"] = df["caste_household"].apply(
        lambda c: c if c in top_10 else "Other"
    )

    # Drop columns
    df.drop(columns=drop_cols, inplace=True)

    return df
```

[13]:
```
VimeoVideo("665415443", h="ca27a7ebfc", width=600)
```

[13]: <IPython.lib.display.VimeoVideo at 0x7f04794d53a0>

Task 4.4.6: Add the query you created in the previous task to the `wrangle` function above. Then import your data by running the cell below. The path to the database is `"/home/jovyan/nepal.sqlite"`.

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

[14]:
```
df = wrangle("/home/jovyan/nepal.sqlite")
df.head(10)
```

[14]:
```
             gender_household_head  age_household_head caste_household  \
household_id
16400201                    Female                46.0        Chhetree
16408101                      Male                66.0        Chhetree
16408901                      Male                54.0           Magar
16409801                      Male                36.0        Chhetree
16410301                    Female                39.0        Chhetree
16418601                    Female                50.0           Sarki
16420401                    Female                48.0           Magar
16420501                    Female                55.0           Magar
16421101                      Male                44.0           Magar
16422001                      Male                46.0           Magar

             education_level_household_head income_level_household  \
household_id
16400201                            Class 5     Rs. 10-20 thousand
16408101                         Illiterate        Rs. 10 thousand
16408901                            Class 4        Rs. 10 thousand
```

16409801	Class 5	Rs. 10 thousand
16410301	Class 4	Rs. 10 thousand
16418601	Illiterate	Rs. 10 thousand
16420401	Intermediate or equivalent	Rs. 10-20 thousand
16420501	Intermediate or equivalent	Rs. 10-20 thousand
16421101	Class 10	Rs. 10 thousand
16422001	Class 6	Rs. 10 thousand

household_id	size_household	is_bank_account_present_in_household \
16400201	4.0	1.0
16408101	5.0	0.0
16408901	5.0	1.0
16409801	6.0	1.0
16410301	3.0	0.0
16418601	5.0	0.0
16420401	4.0	1.0
16420501	4.0	1.0
16421101	5.0	0.0
16422001	6.0	0.0

household_id	age_building	plinth_area_sq_ft	height_ft_pre_eq \
16400201	20	560	18
16408101	21	200	12
16408901	18	315	20
16409801	45	290	13
16410301	21	230	13
16418601	40	250	13
16420401	20	350	13
16420501	45	400	13
16421101	40	250	21
16422001	4	300	12

household_id	land_surface_condition	foundation_type \
16400201	Flat	Mud mortar-Stone/Brick
16408101	Flat	Mud mortar-Stone/Brick
16408901	Flat	Mud mortar-Stone/Brick
16409801	Flat	Mud mortar-Stone/Brick
16410301	Flat	Mud mortar-Stone/Brick
16418601	Flat	Mud mortar-Stone/Brick
16420401	Flat	Mud mortar-Stone/Brick
16420501	Flat	Mud mortar-Stone/Brick
16421101	Moderate slope	Mud mortar-Stone/Brick
16422001	Flat	Mud mortar-Stone/Brick

```
                               roof_type ground_floor_type   other_floor_type  \
household_id
16400201      Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud
16408101      Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud
16408901      Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud
16409801      Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud
16410301      Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud
16418601      Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud
16420401      Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud
16420501      Bamboo/Timber-Heavy roof               Mud  TImber/Bamboo-Mud
16421101      Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud
16422001      Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud

                     position plan_configuration     superstructure  \
household_id
16400201         Not attached        Rectangular  Stone, mud mortar
16408101         Not attached        Rectangular  Stone, mud mortar
16408901         Not attached        Rectangular  Stone, mud mortar
16409801         Not attached        Rectangular  Stone, mud mortar
16410301         Not attached        Rectangular  Stone, mud mortar
16418601         Not attached        Rectangular  Stone, mud mortar
16420401         Not attached        Rectangular  Stone, mud mortar
16420501         Not attached        Rectangular  Stone, mud mortar
16421101      Attached-2 side        Rectangular  Stone, mud mortar
16422001         Not attached        Rectangular  Stone, mud mortar

              vdcmun_id  severe_damage
household_id
16400201             38              0
16408101             38              0
16408901             38              0
16409801             38              0
16410301             38              0
16418601             38              1
16420401             38              1
16420501             38              1
16421101             38              1
16422001             38              1
```

[15]:
```python
# Check your work
assert df.shape == (75883, 20), f"`df` should have shape (75883, 20), not {df.shape}"
```

1.2 Explore

[16]:
```
VimeoVideo("665415463", h="86c306199f", width=600)
```

[16]:
```
<IPython.lib.display.VimeoVideo at 0x7f04794d50d0>
```

Task 4.4.7: 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.

[17]:
```
# Check for high- and low-cardinality categorical features
df.select_dtypes("object").nunique()
```

[17]:
```
gender_household_head             2
caste_household                  11
education_level_household_head   19
income_level_household            5
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
```

[18]:
```
VimeoVideo("665415472", h="1142d69e4a", width=600)
```

[18]:
```
<IPython.lib.display.VimeoVideo at 0x7f04794d5970>
```

[19]:
```
# top_10 = df["caste_household"].value_counts().head(10).index
# df["caste_household"].apply(lambda c: c if c in top_10 else "Other").
 ↪value_counts()
```

Task 4.4.8: Add to your `wrangle` function so that the `"caste_household"` contains only the 10 largest caste groups. For the rows that are not in those groups, `"caste_household"` should be changed to `"Other"`.

- Determine the unique values in a column using pandas.
- Combine multiple categories in a Series using pandas.

[20]:
```
df["caste_household"].nunique()
```

[20]:
```
11
```

```
[21]: # Check your work
assert (
    df["caste_household"].nunique() == 11
), f"The `'caste_household'` column should only have 11 unique values, not␣
↪{df['caste_household'].nunique()}."
```

1.3 Split

```
[22]: VimeoVideo("665415515", h="defc252edd", width=600)
```

```
[22]: <IPython.lib.display.VimeoVideo at 0x7f04794d5af0>
```

Task 4.4.9: Create your feature matrix `X` and target vector `y`. Since our model will only consider building and household data, `X` should not include the municipality column `"vdcmun_id"`. Your target is `"severe_damage"`.

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

```
[24]: # Check your work
assert X.shape == (75883, 18), f"The shape of `X` should be (75883, 18), not {X.
↪shape}."
assert "vdcmun_id" not in X.columns, "There should be no `'vdcmun_id'` column␣
↪in `X`."
assert y.shape == (75883,), f"The shape of `y` should be (75883,), not {y.
↪shape}."
```

Task 4.4.10: 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. Be sure to set a `random_state` for reproducibility.

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

```
[26]: # Check your work
assert X_train.shape == (
    60706,
    18,
), f"The shape of `X_train` should be (60706, 18), not {X_train.shape}."
assert y_train.shape == (
    60706,
), f"The shape of `y_train` should be (60706,), not {y_train.shape}."
assert X_test.shape == (
    15177,
    18,
```

```
), f"The shape of `X_test` should be (15177, 18), not {X_test.shape}."
assert y_test.shape == (
    15177,
), f"The shape of `y_test` should be (15177,), not {y_test.shape}."
```

2 Build Model

2.1 Baseline

Task 4.4.11: Calculate the baseline accuracy score for your model.

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

[27]:
```
acc_baseline = y_train.value_counts(normalize=True).max()
print("Baseline Accuracy:", round(acc_baseline, 2))
```

```
Baseline Accuracy: 0.63
```

2.2 Iterate

Task 4.4.12: Create a Pipeline called `model_lr`. It should have an `OneHotEncoder` transformer and a `LogisticRegression` predictor. Be sure you set the `use_cat_names` argument for your transformer to `True`.

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

[28]:
```
model_lr = make_pipeline(
    OneHotEncoder(use_cat_names=True),
    LogisticRegression(max_iter=3000)
)
model_lr.fit(X_train, y_train)
```

[28]:
```
Pipeline(steps=[('onehotencoder',
                 OneHotEncoder(cols=['gender_household_head', 'caste_household',
                                     'education_level_household_head',
                                     'income_level_household',
                                     '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))])
```

[29]:
```
# Check your work
assert isinstance(
    model_lr, Pipeline
), f"`model_lr` should be a Pipeline, not type {type(model_lr)}."
assert isinstance(
    model_lr[0], OneHotEncoder
), f"The first step in your Pipeline should be a OneHotEncoder, not type␣
↪{type(model_lr[0])}."
assert isinstance(
    model_lr[-1], LogisticRegression
), f"The last step in your Pipeline should be LogisticRegression, not type␣
↪{type(model_lr[-1])}."
check_is_fitted(model_lr)
```

2.3 Evaluate

Task 4.4.13: Calculate the training and test accuracy scores for `model_lr`.

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

[30]:
```
acc_train = accuracy_score(y_train, model_lr.predict(X_train))
acc_test = model_lr.score(X_test, y_test)

print("LR Training Accuracy:", acc_train)
print("LR Validation Accuracy:", acc_test)
```

```
LR Training Accuracy: 0.7181497710275755
LR Validation Accuracy: 0.7220135731699282
```

3 Communicate

[31]:
```
VimeoVideo("665415532", h="00440f76a9", width=600)
```

[31]:
```
<IPython.lib.display.VimeoVideo at 0x7f0471ba4f70>
```

Task 4.4.14: First, extract the feature names and importances from your model. Then create a pandas Series named `feat_imp`, where the index is `features` and the values are your the exponential of the `importances`.

- What's a bar chart?
- Access an object in a pipeline in scikit-learn.
- Create a Series in pandas.

[32]:
```
features = model_lr.named_steps["onehotencoder"].get_feature_names()
importances = model_lr.named_steps["logisticregression"].coef_[0]
feat_imp = pd.Series(np.exp(importances), index=features).sort_values()
feat_imp.head()
```

[32]:
```
superstructure_Brick, cement mortar    0.320384
foundation_type_RC                     0.352191
roof_type_RCC/RB/RBC                   0.413963
ground_floor_type_RC                   0.535611
caste_household_Kumal                  0.540619
dtype: float64
```

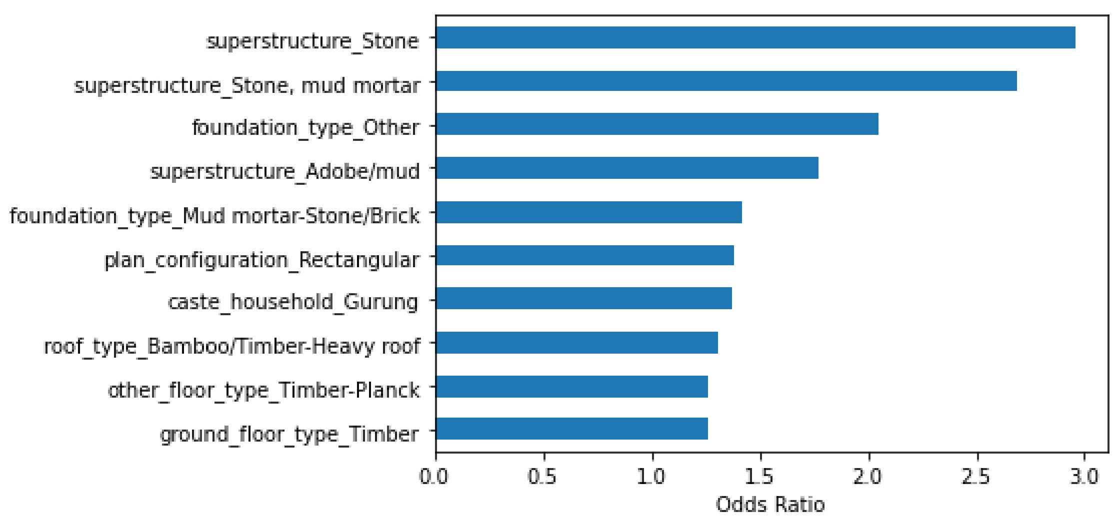
[33]:
```
VimeoVideo("665415552", h="5b2383ccf8", width=600)
```

[33]:
```
<IPython.lib.display.VimeoVideo at 0x7f0471ba4d00>
```

Task 4.4.15: Create a horizontal bar chart with the ten largest coefficients from `feat_imp`. Be sure to label your x-axis `"Odds Ratio"`.

- Create a bar chart using pandas.

[35]:
```
feat_imp.tail(10).plot(kind="barh")
plt.xlabel("Odds Ratio");
```



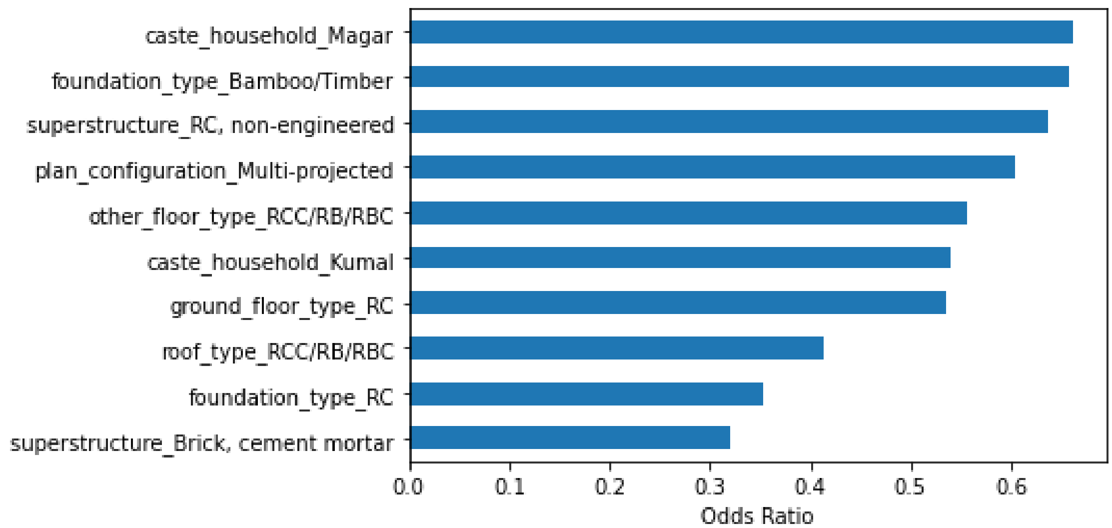
[36]:
```
VimeoVideo("665415581", h="d15477e14d", width=600)
```

[36]:
```
<IPython.lib.display.VimeoVideo at 0x7f0475a5b1f0>
```

Task 4.4.16: Create a horizontal bar chart with the ten smallest coefficients from `feat_imp`. Be sure to label your x-axis `"Odds Ratio"`.

- Create a bar chart using pandas.

[37]:
```
feat_imp.head(10).plot(kind="barh")
plt.xlabel("Odds Ratio");
```



3.1 Explore Some More

[38]:
```
VimeoVideo("665415631", h="90ba264392", width=600)
```

[38]: <IPython.lib.display.VimeoVideo at 0x7f0476c51b20>

Task 4.4.17: Which municipalities saw the highest proportion of severely damaged buildings? Create a DataFrame `damage_by_vdcmun` by grouping `df` by `"vdcmun_id"` and then calculating the mean of the `"severe_damage"` column. Be sure to sort `damage_by_vdcmun` from highest to lowest proportion.

- Aggregate data using the groupby method in pandas.

[41]:
```
damage_by_vdcmun = (df.groupby("vdcmun_id")["severe_damage"].mean().
↪sort_values(ascending=False)).to_frame()
damage_by_vdcmun
```

[41]:

	severe_damage
vdcmun_id	
31	0.930199
32	0.851117
35	0.827145
30	0.824201
33	0.782464
34	0.666979
39	0.572344
40	0.512444
38	0.506425
36	0.503972

```
37                0.437789
```

[42]:
```
# Check your work
assert isinstance(
    damage_by_vdcmun, pd.DataFrame
), f"`damage_by_vdcmun` should be a Series, not type {type(damage_by_vdcmun)}."
assert damage_by_vdcmun.shape == (
    11,
    1,
), f"`damage_by_vdcmun` should be shape (11,1), not {damage_by_vdcmun.shape}."
```

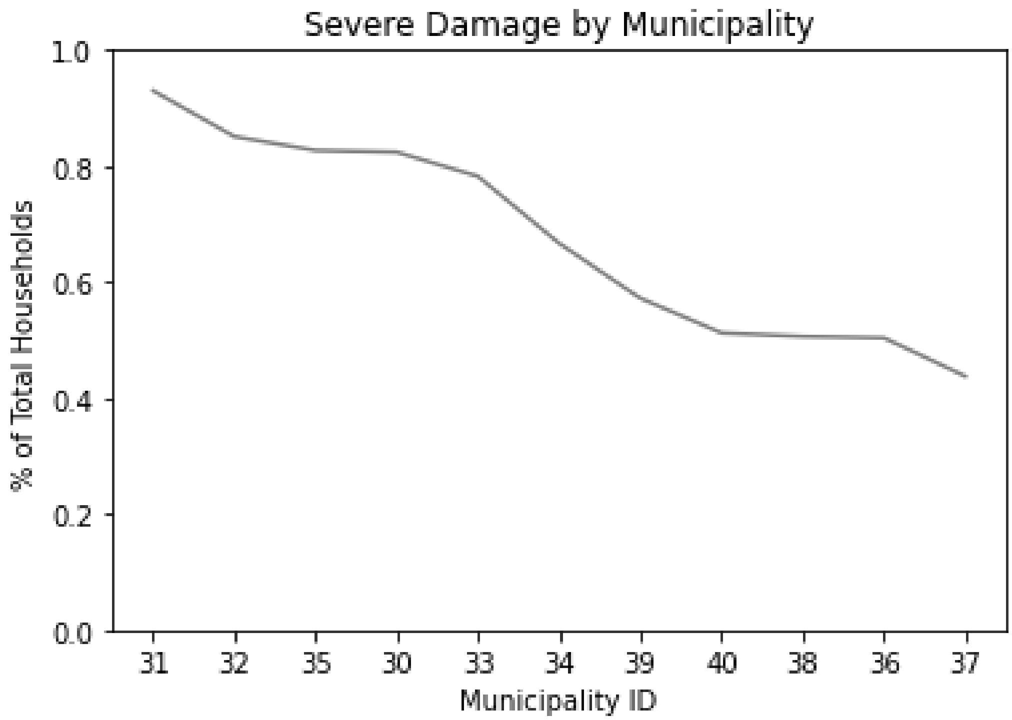
[43]:
```
VimeoVideo("665415651", h="9b5244dec1", width=600)
```

[43]:
```
<IPython.lib.display.VimeoVideo at 0x7f0471ff7ee0>
```

Task 4.4.18: Create a line plot of `damage_by_vdcmun`. Label your x-axis `"Municipality ID"`, your y-axis `"% of Total Households"`, and give your plot the title `"Household Damage by Municipality"`.

- Create a line plot in Matplotlib.

[50]:
```
# Plot line
#damage_by_vdcmun.plot(kind="bar")
plt.plot(damage_by_vdcmun.values, color="grey")
plt.xticks(range(len(damage_by_vdcmun)),labels=damage_by_vdcmun.index)
plt.yticks(np.arange(0.0,1.1,0.2))
plt.xlabel("Municipality ID")
plt.ylabel("% of Total Households")
plt.title("Severe Damage by Municipality");
```



Given the plot above, our next question is: How are the Gurung and Kumal populations distributed across these municipalities?

[51]:
```
VimeoVideo("665415693", h="fb2e54aa04", width=600)
```

[51]:
```
<IPython.lib.display.VimeoVideo at 0x7f047197a070>
```

Task 4.4.19: Create a new column in `damage_by_vdcmun` that contains the the proportion of Gurung households in each municipality.

- Aggregate data using the groupby method in pandas.
- Create a Series in pandas.

[58]:
```
damage_by_vdcmun["Gurung"] = (
    df[df["caste_household"] == "Gurung"].groupby("vdcmun_id")["severe_damage"].
↪count()
    / df.groupby("vdcmun_id")["severe_damage"].count()
)
damage_by_vdcmun
```

[58]:
```
           severe_damage    Gurung
vdcmun_id
31              0.930199  0.326937
32              0.851117  0.387849
35              0.827145  0.826889
```

```
30              0.824201  0.338152
33              0.782464  0.011943
34              0.666979  0.385084
39              0.572344  0.097971
40              0.512444  0.246727
38              0.506425  0.049023
36              0.503972  0.143178
37              0.437789  0.050485
```

[59]:
```
VimeoVideo("665415707", h="9b29c23434", width=600)
```

[59]:
```
<IPython.lib.display.VimeoVideo at 0x7f047197a8e0>
```

Task 4.4.20: Create a new column in `damage_by_vdcmun` that contains the the proportion of Kumal households in each municipality. Replace any `NaN` values in the column with `0`.

- Aggregate data using the groupby method in pandas.
- Create a Series in pandas.

[61]:
```
damage_by_vdcmun["Kumal"] = (
    df[df["caste_household"] == "Kumal"].groupby("vdcmun_id")["severe_damage"].
 ↪count()
    / df.groupby("vdcmun_id")["severe_damage"].count()
).fillna(0)
damage_by_vdcmun
```

[61]:
```
           severe_damage    Gurung     Kumal
vdcmun_id
31              0.930199  0.326937  0.000000
32              0.851117  0.387849  0.000000
35              0.827145  0.826889  0.000000
30              0.824201  0.338152  0.000000
33              0.782464  0.011943  0.029478
34              0.666979  0.385084  0.000000
39              0.572344  0.097971  0.000267
40              0.512444  0.246727  0.036973
38              0.506425  0.049023  0.100686
36              0.503972  0.143178  0.003282
37              0.437789  0.050485  0.048842
```

[62]:
```
VimeoVideo("665415729", h="8d0712c306", width=600)
```

[62]:
```
<IPython.lib.display.VimeoVideo at 0x7f047197a0a0>
```

Task 4.4.21: Create a visualization that combines the line plot of severely damaged households you made above with a stacked bar chart showing the proportion of Gurung and Kumal households in each district. Label your x-axis `"Municipality ID"`, your y-axis `"% of Total Households"`.

- Create a bar chart using pandas.

- Drop a column from a DataFrame using pandas.

[67]:
```
damage_by_vdcmun.drop(columns="severe_damage").plot(
    kind = "bar", stacked=True
)

plt.plot(damage_by_vdcmun["severe_damage"].values, color="grey")
plt.xticks(range(len(damage_by_vdcmun)),labels=damage_by_vdcmun.index)
plt.yticks(np.arange(0.0,1.1,0.2))
plt.xlabel("Municipality ID")
plt.ylabel("% of Total Households")
plt.title("Severe Damage by Municipality");
```