DATA SOCIETY®

Advanced classification - day 4

"One should look for what is and not what he thinks should be."
-Albert Einstein.

Module completion checklist

Objective	Complete
Summarize feature engineering and its usefulness	
Explain the feature engineering of numerical variables	
Implement feature engineering of the numerical variable on Costa Rican dataset	
Derive additional features on Costa Rican dataset	
Explain the feature engineering of categorical variables	
Implement feature engineering of the categorical variable on Costa Rican dataset	
Handle temporal and spatial predictors in the data	
Explain the principle of parsimony and feature selection	

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- Let the `main_dir` be the variable corresponding to your `af-werx` folder

```
# Set `main_dir` to the location of your `af-werx` folder (for Linux).
main_dir = "/home/[username]/Desktop/af-werx"
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Mac).
main_dir = '/Users/[username]/Desktop/af-werx'
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Windows).
main_dir = "C:\\Users\\[username]\\Desktop\\af-werx"
```

```
# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory.
data_dir = main_dir + "/data"
```

Loading packages

Let's load the packages we will be using:

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.model_selection import train_test_split
import warnings
import datetime as dt
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Working directory

- Set working directory to the `data_dir` variable we set
- We do this using the `os.chdir` function, change directory
- We can then check the working directory using `.getcwd()`
- For complete documentation of the `os` package, ***click here***

```
# Set working directory.
os.chdir(data_dir)
```

```
# Check working directory.
print(os.getcwd())
```

```
/home/[user-name]/Desktop/af-werx/data
```

What have we mastered so far?

- Loading and preparing our data
- Finding target and predictors
- Modeling to predict the target using our predictors (the predictors are also known as features)
- **What if we can add more predictors to our model so that we can predict better?**
- **What if we can derive additional predictors from our existing data and improve our model performance?**
- **What if we can reduce the number of features, but maintain performance?**

What is feature engineering?

- Feature engineering is the process of formulating the **most appropriate features to improve our model**
- It is the process of using **domain knowledge** of the data to create features that make machine learning algorithms work
- The **quality and quantity** of the features are very important as they influence the results
- Today we will see numerous techniques of feature engineering to our data
- Remember **not all techniques are effective on all datasets**

Datasets for today!

- **We will be using the following datasets. We discussed each of the datasets and use cases already**

- **One dataset to learn the concepts in class**
  - **Costa Rica household poverty data**
  - **Room occupancy data** - to feature engineer date variables

- **One dataset for our in-class exercises**
  - **Community and crime dataset**
  - **Invoice risk detection data** - to feature engineer date variables

Today's task

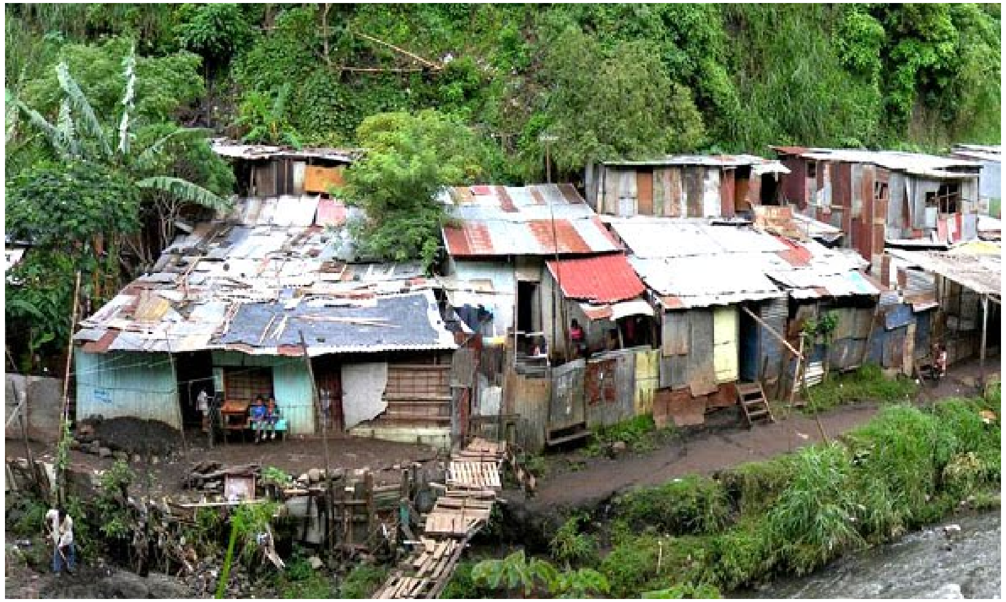
- We will be using the Costa Rica dataset for most parts of the module
- We will check if our feature engineering techniques work on our dataset as follows:
  - We will initially build a baseline model using **logistic regression**
  - We will **add features** to our dataset using different techniques
  - Then we will **train using logistic regression** after additional features are added
  - We will **compare the performance** with the baseline model and derive insights

Costa Rican poverty: proposed solution

Costa Rican poverty level prediction: proposed solution

- To improve on PMT, the IDB built a competition for Kaggle participants to use methods beyond traditional econometrics

- The given dataset contains Costa Rican household characteristics with a target of four categories:
  - extreme poverty
  - moderate poverty
  - vulnerable households
  - non vulnerable households



Load the dataset

- Load the Costa Rican dataset and print the head

```
costa_rica = pd.read_csv("costa_rica_poverty.csv")
costa_rica.head()
```

```
  household_id         ind_id  rooms  ...  age  Target  monthly_rent
0    21eb7fcc1  ID_279628684      3  ...   43       4      190000.0
1    0e5d7a658  ID_f29eb3ddd      4  ...   67       4      135000.0
2    2c7317ea8  ID_68de51c94      8  ...   92       4           NaN
3    2b58d945f  ID_d671db89c      5  ...   17       4      180000.0
4    2b58d945f  ID_d56d6f5f5      5  ...   37       4      180000.0

[5 rows x 84 columns]
```

Subset the data for baseline model

- Let's subset the data and check for NAs to build the baseline model

```
costa_subset = costa_rica[['rooms', 'males_tot',
'females_tot', 'years_of_schooling',
'num_child', 'num_adults', 'num_65plus',
                           'dependency_rate',
'male_hh_head_educ', 'female_hh_head_educ',
'meaneduc',
                           'bedrooms',
'ppl_per_room', 'num_mobilephones', 'age',
'Target']]
```

```
costa_subset.shape
```

```
(9557, 16)
```

```
costa_subset.isnull().sum()
```

```
rooms                  0
males_tot              0
females_tot            0
years_of_schooling     0
num_child              0
num_adults             0
num_65plus             0
dependency_rate        0
male_hh_head_educ      0
female_hh_head_educ    0
meaneduc               0
bedrooms               0
ppl_per_room           0
num_mobilephones       0
age                    0
Target                 0
dtype: int64
```

Create target and predictors

- Let's create our target as binary and split the predictors and target
- Here, we'll say that any household that is classified as 3 or below for household poverty will be labeled as 'vulnerable', while any household with classified as 4 will be labeled 'non vulnerable'

```
costa_subset['Target_binary'] = np.where(costa_subset['Target'] <= 3, 'vulnerable', 'non_vulnerable')
```

```
X = costa_subset.drop(['Target', 'Target_binary'], axis = 1)
y = costa_subset.Target_binary
```

Logistic regression function

- Today, we will be building a lot of models and comparing them
- So instead of training, fitting and finding the accuracy every time, let's go ahead and create a function which does all the tasks and returns the accuracy value for our comparison

```
def logistic_model(X,y):
    np.random.seed(1)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30)
    logreg = LogisticRegression()
    logreg.fit(X_train, y_train)
    y_pred = logreg.predict(X_test)
    accuracy = metrics.accuracy_score(y_test, y_pred)
    return(accuracy)
```

Create the baseline and save the result

- Let's call the function to create the baseline model

```
accuracy1 = logistic_model(X,y)
accuracy1
```

```
0.7629009762900977
```

- We will create a performance dataframe to save the accuracy values

```
performance_df = pd.DataFrame(columns = ['dataset_name', 'model_name', 'model_metric', 'metric_value'])

s = pd.Series(['Costa_rica', 'baseline_model', 'accuracy', accuracy1], index = ['dataset_name',
'model_name', 'model_metric', 'metric_value'])
performance_df = performance_df.append(s, ignore_index = True)
performance_df
```

	dataset_name	model_name	model_metric	metric_value
0	Costa_rica	baseline_model	accuracy	0.762901

Feature engineering of numerical variables

- There are numerous ways we can do feature engineering for numerical variables
  - Check for data consistency and handle junk
  - Outlier handling
  - Scaling the features
  - Best practices for imputing NA
  - Binning/discretization
  - Adding additional features based on domain knowledge
- We have spoken about few of them already in our previous modules!

Check for data consistency and handle junk

- There are some important first steps for feature engineering:
- **Check** whether all the **predictors** we read from the csv file are in correct format
- **Check** for **data types**
- Make sure that there are **no error values**, such as having text data instead of numeric values
- Check that the **features are in correct format**

Imputing NA in numerical variables

- By far, we have seen that we usually impute the missing values with **mean**
- But should we always impute the missing values with mean?
- There are other ways as well:
  - If the variable has more than 50%-70% NAs, then it is safe to ignore the variable while modeling
  - If the variable has a normal distribution, then the data is symmetric so it makes sense to impute with mean
  - If the variable is skewed, then we can impute the data with median

Using supervised learning models to impute NAs

- Sometimes we can use machine learning models to impute NAs as well
- Separate the rows containing missing values as the test data
- Train the model using the remaining rows which contains non NA values
- Predict on the test data
- We can use any of the machine learning technique we have seen so far, such as:
  - Linear regression
  - Decision trees for regression
  - kNN for regression

Knowledge check 1

Exercise 1

Module completion checklist

Objective	Complete
Summarize feature engineering and its usefulness	✔
Explain the feature engineering of numerical variables	✔
Implement feature engineering of the numerical variable on Costa Rican dataset	
Derive additional features on Costa Rican dataset	
Explain the feature engineering of categorical variables	
Implement feature engineering of the categorical variable on Costa Rican dataset	
Handle temporal and spatial predictors in the data	
Explain the principle of parsimony and feature selection	

Imputation using regression

- We saw that monthly rent has more than 50% NAs in our previous module
- It might make more sense to ignore the variable, but we will use two imputation techniques and compare with our baseline and decide
- What if we impute NAs in `monthly_rent` using linear regression?

```
# Subset the data once again.
costa_regression_subset = costa_rica[['rooms','males_tot', 'females_tot', 'years_of_schooling',
'num_child', 'num_adults', 'num_65plus',
                            'dependency_rate', 'male_hh_head_educ', 'female_hh_head_educ', 'meaneduc',
                            'bedrooms', 'ppl_per_room', 'num_mobilephones', 'age', 'monthly_rent',
'Target']]

# Create target variable.
costa_regression_subset['Target_binary'] = np.where(costa_regression_subset['Target'] <= 3,
'vulnerable', 'non_vulnerable')
```

Imputation using regression

- Split training data as the rows that contain numerical monthly rent value and test data as the rows that contain NA in monthly rent
- Monthly rent is going to be our target

```
train_subset = costa_regression_subset[~costa_regression_subset['monthly_rent'].isna()]
test_subset = costa_regression_subset[costa_regression_subset['monthly_rent'].isna()]

print(train_subset.shape)
```

```
(2697, 18)
```

```
print(test_subset.shape)
```

```
(6860, 18)
```

Imputation using regression

- Separate target and predictors and fit the model with monthly rent as the target

```
X_reg_train = train_subset.drop(['Target', 'Target_binary', 'monthly_rent'], axis = 1)
y_reg_train = train_subset.monthly_rent
X_reg_test = test_subset.drop(['Target', 'Target_binary', 'monthly_rent'], axis = 1)

X_reg_train.shape
```

```
(2697, 15)
```

```
# Fit the model and predict on test
linear_model = LinearRegression()
linear_model.fit(X_reg_train, y_reg_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
y_pred = linear_model.predict(X_reg_test)
len(y_pred)
```

```
6860
```

Imputation using regression

- Impute the predicted monthly rent value for NA values

```
# Impute the monthly rent with predicted target value.
test_subset['monthly_rent'] = y_pred

# Concatenate training and test data.
```

```
imputed_df = pd.concat([train_subset, test_subset])

# Split X and y to predict our poverty level now.
imputed_X = imputed_df.drop(['Target', 'Target_binary'], axis = 1)
imputed_y = imputed_df.Target_binary
```

Model performance after imputation

- Find the performance of logistic regression with the imputed monthly rent as one of the predictors

```
accuracy2 = logistic_model(imputed_X,imputed_y)
print(accuracy2)
```

```
0.648186889818689
```

```
s = pd.Series(['Costa_rica', 'na_imputation_regression', 'accuracy', accuracy2], index =
['dataset_name', 'model_name', 'model_metric', 'metric_value'])
performance_df = performance_df.append(s, ignore_index = True)
performance_df
```

	dataset_name	model_name	model_metric	metric_value
0	Costa_rica	baseline_model	accuracy	0.762901
1	Costa_rica	na_imputation_regression	accuracy	0.648187

- The accuracy has decreased when imputed with regression, so it's not a good technique for our dataset

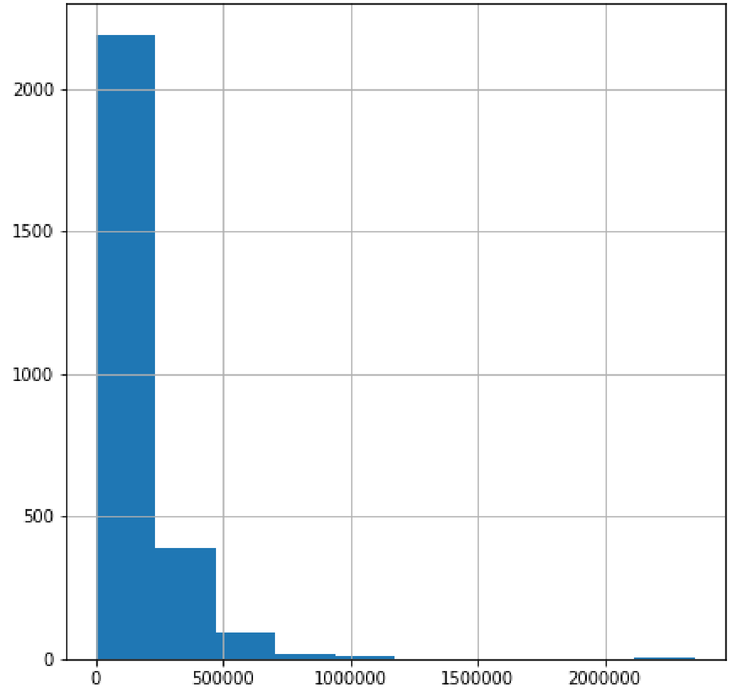
Why is regression not good technique?

- Why do you think that imputation of NA with regression decreased our performance?
- It could be because we are trying to impute 6000 records with a model made of 2000 records
- More than 50% NA is always a problem
- But we should also check if the assumptions of linear regression are satisfied or not

Plot monthly rent

- Check the distribution of monthly rent

```
# Check the distribution of the data
costa_rica['monthly_rent'].hist()
```



- Monthly rent is left skewed
- It is not normally distributed, hence it makes more sense to impute with median rather than mean

NA imputation with median

```
costa_rica['monthly_rent'].isnull().value_counts()
```

```
True     6860
False    2697
Name: monthly_rent, dtype: int64
```

- Impute the NA with median as the dataset looks skewed to the left

```
costa_rica['monthly_rent'].fillna(costa_rica['monthly_rent'].median(), inplace = True)
X = pd.concat([X, costa_rica['monthly_rent']], axis = 1)
```

Model on NA imputed data

```
accuracy3 = logistic_model(X,y)
accuracy3
```

```
0.6328451882845189
```

```
s = pd.Series(['Costa_rica', 'na_imputation_median', 'accuracy', accuracy3], index=['dataset_name',
'model_name', 'model_metric', 'metric_value'])
performance_df = performance_df.append(s, ignore_index = True)
performance_df
```

	dataset_name	model_name	model_metric	metric_value
0	Costa_rica	baseline_model	accuracy	0.762901
1	Costa_rica	na_imputation_regression	accuracy	0.648187
2	Costa_rica	na_imputation_median	accuracy	0.632845

- Our accuracy decreased, so imputing with median isn't a good idea either
- It is better to drop the variable as the predictor from our subset since it decreases our performance

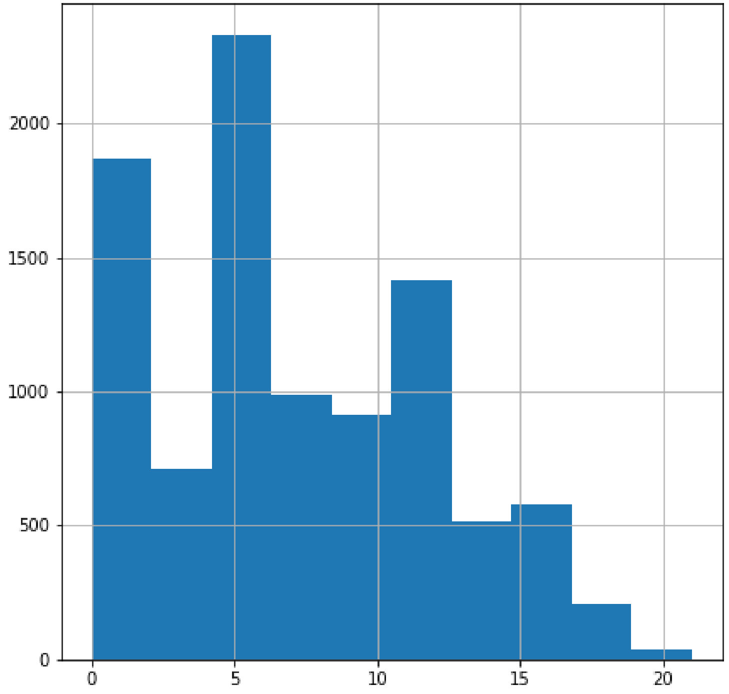
Binning numeric variables

- We can **bin the numeric variables** into categories and add the binned variable as an additional feature
- The main aim behind binning is to make your model more robust and avoid overfitting
- Let's see how binned feature can be used in models:
  - Say we want to predict if it will rain tomorrow
  - We have the temperature in Fahrenheit
  - Binning temperature to low/high instead of the actual temperature makes more sense as it is more likely to rain at low temperature and the actual numerical value can be avoided

Plot years of schooling

- Let's see the distribution of `years_of_schooling`

```
X['years_of_schooling'].hist()
```



Binning a variable in our data

- We are going to bin the variable as `less_than_10` and `more_than_10`

```
# Bin the variables.
X['school_category'] = pd.cut(X.age, [-1,10,25], labels = ["less_than_10", "more_than_10"])

# Add the school category as one of the variables.
X['school_category'] = X['school_category'].cat.codes
school_cat = pd.get_dummies(X['school_category'], prefix = 'school', drop_first = True)
X = pd.concat([X, school_cat], axis = 1)

# Drop the monthly rent and years of schooling variables.
X.drop(['years_of_schooling', 'monthly_rent', 'school_category'], axis = 1, inplace = True)
X.columns
```

```
Index(['rooms', 'males_tot', 'females_tot', 'num_child', 'num_adults',
       'num_65plus', 'dependency_rate', 'male_hh_head_educ',
       'female_hh_head_educ', 'meaneduc', 'bedrooms', 'ppl_per_room',
       'num_mobilephones', 'age', 'school_0', 'school_1'],
      dtype='object')
```

Model performance after binning

```
accuracy4 = logistic_model(X,y)
print(accuracy4)
```

```
0.7670850767085077
```

```
s = pd.Series(['Costa_rica', 'binning', 'accuracy', accuracy4], index=['dataset_name', 'model_name', 'model_metric', 'metric_value'])
performance_df = performance_df.append(s, ignore_index = True)
performance_df
```

	dataset_name	model_name	model_metric	metric_value
0	Costa_rica	baseline_model	accuracy	0.762901
1	Costa_rica	na_imputation_regression	accuracy	0.648187
2	Costa_rica	na_imputation_median	accuracy	0.632845
3	Costa_rica	binning	accuracy	0.767085

- When compared to our baseline model, the accuracy improved a bit

Knowledge check 2

Exercise 2

Module completion checklist

Objective	Complete
Summarize feature engineering and its usefulness	✔
Explain the feature engineering of numerical variables	✔
Implement feature engineering of the numerical variable on Costa Rican dataset	✔
Derive additional features on Costa Rican dataset	
Explain the feature engineering of categorical variables	
Implement feature engineering of the categorical variable on Costa Rican dataset	
Handle temporal and spatial predictors in the data	
Explain the principle of parsimony and feature selection	

Adding more features

- Sometimes deriving new features from the existing features can improve the performance considerably
- It requires more domain knowledge on the dataset
- For example, consider we have the birth dates of patients and we want to predict whether the patient has cancer or not
- Deriving age from the birth date and adding that as a feature would give us better prediction than keeping it as birth date because age is an important factor for cancer according to clinical research
- Now, we will add features in our Costa Rica dataset and check how it performs on the data

Additional features on Costa Rica dataset

- Let's understand the background of our data before adding more features
- We are trying to predict household level poverty
- But each row contributes to each individual of household
- There is `household_id` which is the unique identifier of each household and `ind_id` which is unique for each row (individual in our data)
- Members of the same household have the same `household_id`
- Do you think adding some household_level to each row can make our model better?
- We are going to build 3 models after adding a few household level features

Additional features for model 1

- The variables `married`, `separated` and `single` are binary in each row
- It tells if each person is married/not, separated/not or single/not
- We are going to find total number of married, separated and single members of each household and add this as a feature and check our model performance

```
cat_count_subset = costa_rica[['household_id', 'married', 'separated', 'single']]
cat_count_subset.head()
```

```
  household_id  married  separated  single
0    21eb7fcc1        0          0       0
1    0e5d7a658        0          0       0
2    2c7317ea8        0          0       0
3    2b58d945f        0          0       1
4    2b58d945f        0          0       0
```

- Use `groupby` to group data by household and find the total number of members

```
count_grouped = cat_count_subset.groupby(['household_id']).sum()
count_grouped.reset_index(inplace = True)
```

View the additional features

```
count_grouped.head()
```

```
  household_id  married  separated  single
0    001ff74ca        0          1       0
1    003123ec2        0          0       0
2    004616164        0          0       1
3    004983866        0          0       1
4    005905417        0          1       1
```

```
# Add the features to our model subset.
X = pd.concat([X, costa_rica['household_id']], axis = 1)          #<- add household id to X to find the match
X = pd.merge(X, count_grouped, how = 'left', on = 'household_id') #<- merge on household id
X.drop(['household_id'], axis = 1, inplace = True)                #<- drop household id
X.columns
```

```
Index(['rooms', 'males_tot', 'females_tot', 'num_child', 'num_adults',
       'num_65plus', 'dependency_rate', 'male_hh_head_educ',
       'female_hh_head_educ', 'meaneduc', 'bedrooms', 'ppl_per_room',
       'num_mobilephones', 'age', 'school_0', 'school_1', 'married',
       'separated', 'single'],
      dtype='object')
```

Additional feature model 1

```
accuracy5 = logistic_model(X, y)
print(accuracy5)
```

```
0.7691771269177127
```

```
s = pd.Series(['Costa_rica', 'categorical_summary', 'accuracy', accuracy5], index=['dataset_name',
'model_name', 'model_metric', 'metric_value'])
performance_df = performance_df.append(s, ignore_index = True)
performance_df
```

	dataset_name	model_name	model_metric	metric_value
0	Costa_rica	baseline_model	accuracy	0.762901
1	Costa_rica	na_imputation_regression	accuracy	0.648187
2	Costa_rica	na_imputation_median	accuracy	0.632845
3	Costa_rica	binning	accuracy	0.767085
4	Costa_rica	categorical_summary	accuracy	0.769177

- Looks like these features improved the model by a little

Additional features for model 2

- Let's add the summary statistics at household level to individual data and check the performance
- Take `age` and `years_of_schooling` and find the summary of the data at household level
- Add the summary details to our model data

```
numerical_summary_subset = costa_rica[['household_id', 'years_of_schooling', 'age']]

numerical_summary_subset = numerical_summary_subset.groupby(['household_id']).agg(['min', 'max',
'mean', 'sum'])

numerical_summary_subset.columns = ['school_min', 'school_max', 'school_mean', 'school_sum',
                                    'age_min', 'age_max', 'age_mean', 'age_sum']

numerical_summary_subset.reset_index(inplace = True)

numerical_summary_subset.head()
```

```
  household_id  school_min  school_max  ...  age_max   age_mean  age_sum
0    001ff74ca           0          16  ...       38  19.000000       38
1    003123ec2           0           7  ...       24  12.750000       51
2    004616164           3          11  ...       50  33.000000       66
3    004983866           7           8  ...       59  37.500000       75
4    005905417           0           9  ...       32  17.333333       52

[5 rows x 9 columns]
```

Add the additional features

- Add the new features to our model data and remove the features which we added previously

```
X = pd.concat([X, costa_rica['household_id']], axis = 1)
X = pd.merge(X, numerical_summary_subset, how = 'left', on = 'household_id')
X.drop(['household_id', 'married', 'separated', 'single'], axis = 1, inplace = True) #<- drop previous features
X.columns
```

```
Index(['rooms', 'males_tot', 'females_tot', 'num_child', 'num_adults',
       'num_65plus', 'dependency_rate', 'male_hh_head_educ',
       'female_hh_head_educ', 'meaneduc', 'bedrooms', 'ppl_per_room',
       'num_mobilephones', 'age', 'school_0', 'school_1', 'school_min',
       'school_max', 'school_mean', 'school_sum', 'age_min', 'age_max',
       'age_mean', 'age_sum'],
      dtype='object')
```

Additional feature model 2

```
accuracy6 = logistic_model(X,y)
print(accuracy6)
```

```
0.7744072524407253
```

```
s = pd.Series(['Costa_rica', 'numerical_summary', 'accuracy', accuracy6], index=['dataset_name',
'model_name', 'model_metric', 'metric_value'])
performance_df = performance_df.append(s, ignore_index = True)
performance_df
```

	dataset_name	model_name	model_metric	metric_value
0	Costa_rica	baseline_model	accuracy	0.762901
1	Costa_rica	na_imputation_regression	accuracy	0.648187
2	Costa_rica	na_imputation_median	accuracy	0.632845
3	Costa_rica	binning	accuracy	0.767085
4	Costa_rica	categorical_summary	accuracy	0.769177
5	Costa_rica	numerical_summary	accuracy	0.774407

- Our model has improved slightly with our summary statistics features

Additional feature model 3

- We are going to add 5 additional features with 4 as `male_per_room`, `female_per_room`, `room_per_person`, `bed_per_person`
- These are all features derived from our existing number of male, female, total people, room and bedroom

```
costa_rica['male_per_room'] = costa_rica['males_tot'] / costa_rica['rooms']
costa_rica['female_per_room'] = costa_rica['females_tot'] / costa_rica['rooms']
costa_rica['room_per_person'] = costa_rica['rooms'] / costa_rica['ppl_total']
costa_rica['bed_per_room'] = costa_rica['bedrooms'] / costa_rica['rooms']
```

Additional feature model 3

- The 5th feature is the housing condition which we derive as follows
- We have:
  - Wall conditions - wall_bad, wall_reg or wall_good
  - Roof conditions - roof_bad, roof_reg, roof_good
  - Floor conditions - floor_bad, floor_reg, floor_good
- We will rank wall condition bad, regular and good as 0, 1 and 2
- We sum up the wall condition and similarly we sum up roof and floor condition as well
- So overall, a person having higher housing condition value (highest possible value is 6) has a good wall, roof and floor

Additional feature model 3

```
costa_rica['wall'] = np.argmax(np.array(costa_rica[['wall_bad', 'wall_reg', 'wall_good']]), axis = 1)
costa_rica['roof'] = np.argmax(np.array(costa_rica[['roof_bad', 'roof_reg', 'roof_good']]), axis = 1)
costa_rica['floor'] = np.argmax(np.array(costa_rica[['floor_bad', 'floor_reg', 'floor_good']]), axis =
1)
costa_rica['house_condition'] = costa_rica['wall'] + costa_rica['roof'] + costa_rica['floor']
costa_rica[['wall', 'roof', 'floor', 'house_condition']].head()
```

	wall	roof	floor	house_condition
0	1	0	0	1
1	1	1	1	3
2	1	2	2	5
3	2	2	2	6
4	2	2	2	6

- Add the new 5 features to the model data

```
X = pd.concat([X, costa_rica['house_condition'], costa_rica['male_per_room'],
costa_rica['female_per_room'],
               costa_rica['room_per_person'], costa_rica['bed_per_room']], axis = 1)
```

Additional feature model 3

- Let's build another new model with our new added features

```
accuracy7 = logistic_model(X,y)
print(accuracy7)
```

```
0.7705718270571827
```

```
s = pd.Series(['Costa_rica', 'additional_features', 'accuracy', accuracy7], index=['dataset_name',
'model_name', 'model_metric', 'metric_value'])
performance_df = performance_df.append(s, ignore_index = True)
performance_df
```

	dataset_name	model_name	model_metric	metric_value
0	Costa_rica	baseline_model	accuracy	0.762901
1	Costa_rica	na_imputation_regression	accuracy	0.648187
2	Costa_rica	na_imputation_median	accuracy	0.632845
3	Costa_rica	binning	accuracy	0.767085
4	Costa_rica	categorical_summary	accuracy	0.769177
5	Costa_rica	numerical_summary	accuracy	0.774407
6	Costa_rica	additional_features	accuracy	0.770572

- Our accuracy decreased after new features were added

Module completion checklist

Objective	Complete
Summarize feature engineering and its usefulness	✔
Explain the feature engineering of numerical variables	✔
Implement feature engineering of the numerical variable on Costa Rican dataset	✔
Derive additional features on Costa Rican dataset	✔
Explain the feature engineering of categorical variables	✔
Implement feature engineering of the categorical variable on Costa Rican dataset	
Handle temporal and spatial predictors in the data	
Explain the principle of parsimony and feature selection	

Feature engineering of categorical variables

- A general method is to convert the categorical to dummy variables and use them in the model
- If we have a lot of categories in the column, we can keep the most occurring categories as distinct categories and keep all the low frequency categories as `other` category
- If there is a missing value in a categorical variable
  - Impute with most occurring category if more than 60% of the column has that category
  - We can also use classification models to impute the NAs as we did for the numerical column using regression

Feature engineering in Costa Rica data

- We already saw how to convert our categorical variables as dummy
- Let's add some dummy variables and check our result

```
dummy_columns = ['urban_zone', 'rural_zone', 'tablet', 'house_rented', 'computer', 'television']

for i in range(0, len(dummy_columns)):
    colname = "df_" + str(i)
    costa_rica[dummy_columns[i]] = pd.Categorical(costa_rica[dummy_columns[i]])
    costa_rica[dummy_columns[i]] = costa_rica[dummy_columns[i]].cat.codes
    colname = pd.get_dummies(costa_rica[dummy_columns[i]], prefix = (str(dummy_columns[i])), drop_first
= True)
    X = pd.concat([X, colname], axis = 1)
```

View the features

```
X = X.drop(['house_condition', 'male_per_room', 'female_per_room', 'room_per_person', 'bed_per_room'],
axis = 1)

X.columns
```

```
Index(['rooms', 'males_tot', 'females_tot', 'num_child', 'num_adults',
       'num_65plus', 'dependency_rate', 'male_hh_head_educ',
       'female_hh_head_educ', 'meaneduc', 'bedrooms', 'ppl_per_room',
       'num_mobilephones', 'age', 'school_0', 'school_1', 'school_min',
       'school_max', 'school_mean', 'school_sum', 'age_min', 'age_max',
       'age_mean', 'age_sum', 'urban_zone_1', 'rural_zone_1', 'tablet_1',
       'house_rented_1', 'computer_1', 'television_1'],
      dtype='object')
```

Model after adding dummy variables

```
accuracy8 = logistic_model(X, y)
print(accuracy8)
```

```
0.7782426778242678
```

```
s = pd.Series(['Costa_rica', 'dummy_encoding', 'accuracy', accuracy8], index = ['dataset_name',
'model_name', 'model_metric', 'metric_value'])
performance_df = performance_df.append(s, ignore_index = True)
performance_df
```

	dataset_name	model_name	model_metric	metric_value
0	Costa_rica	baseline_model	accuracy	0.762901
1	Costa_rica	na_imputation_regression	accuracy	0.648187
2	Costa_rica	na_imputation_median	accuracy	0.632845
3	Costa_rica	binning	accuracy	0.767085
4	Costa_rica	categorical_summary	accuracy	0.769177
5	Costa_rica	numerical_summary	accuracy	0.774407
6	Costa_rica	additional_features	accuracy	0.770572
7	Costa_rica	dummy_encoding	accuracy	0.778243

- Our accuracy improved after adding the dummy variables
- The last model has the highest accuracy so we can say it performs better in predicting our target after feature engineering and we choose that as our final model

Other ways of feature engineering

- We can use unsupervised learning methods to reduce our number of features but improve our performance
- Take the numerical subset of the data and run the k means algorithm
- Add the cluster values instead of the numerical subset
- We already saw how to use PCA to improve our performance as well

Knowledge check 3

Exercise 3

Module completion checklist

Objective	Complete
Summarize feature engineering and its usefulness	✔
Explain the feature engineering of numerical variables	✔
Implement feature engineering of the numerical variable on Costa Rican dataset	✔
Derive additional features on Costa Rican dataset	✔
Explain the feature engineering of categorical variables	✔
Implement feature engineering of the categorical variable on Costa Rican dataset	✔
Handle temporal and spatial predictors in the data	
Explain the principle of parsimony and feature selection	

Feature engineering of date variables

- Sometimes our datasets might contain a date variable
- How do we handle date variables?
- We can extract features from the date variable as follows:
  - Day
  - Year
  - Month
  - Hour
  - Week number
  - Weekday/weekend as categorical variable
  - Bin hours as peak hour/non peak hour
- Use your best judgment to understand the data and add features which make sense

Data set for date features

- We will use the room occupancy dataset to do feature engineering for date variables as our Costa Rica dataset does not contain any date variable
- This dataset classifies whether the room is occupied, or not, based on the features
- Load the dataset

```
occupancy_data = pd.read_csv('occupancy_data.csv')
occupancy_data.head()
```

	date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
0	2/4/15 17:51	23.18	27.2720	426.0	721.25	0.004793	1
1	2/4/15 17:51	23.15	27.2675	429.5	714.00	0.004783	1
2	2/4/15 17:53	23.15	27.2450	426.0	713.50	0.004779	1
3	2/4/15 17:54	23.15	27.2000	426.0	708.25	0.004772	1
4	2/4/15 17:55	23.10	27.2000	426.0	704.50	0.004757	1

Data set for date features

```
occupancy_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8143 entries, 0 to 8142
Data columns (total 7 columns):
date             8143 non-null object
Temperature      8143 non-null float64
Humidity         8143 non-null float64
Light            8143 non-null float64
CO2              8143 non-null float64
HumidityRatio    8143 non-null float64
Occupancy        8143 non-null int64
dtypes: float64(5), int64(1), object(1)
memory usage: 445.4+ KB
```

- Occupancy 1 specifies that the room is occupied and 0 means it is not occupied

Baseline model for comparison

- Let's first create a baseline model without the date variable to compare our model

```
occupancy_X = occupancy_data.drop(['date', 'Occupancy'], axis = 1)
occupancy_y = occupancy_data.Occupancy

accuracy9 = logistic_model(occupancy_X, occupancy_y)
print(accuracy9)
```

```
0.9897666803110929
```

```
s = pd.Series(['Occupancy', 'occupancy_baseline', 'accuracy', accuracy9], index = ['dataset_name',
'model_name', 'model_metric', 'metric_value'])
performance_df = performance_df.append(s, ignore_index = True)
performance_df
```

```
  dataset_name                model_name model_metric  metric_value
0   Costa_rica            baseline_model     accuracy      0.762901
1   Costa_rica  na_imputation_regression     accuracy      0.648187
2   Costa_rica      na_imputation_median     accuracy      0.632845
3   Costa_rica                   binning     accuracy      0.767085
4   Costa_rica       categorical_summary     accuracy      0.769177
5   Costa_rica         numerical_summary     accuracy      0.774407
6   Costa_rica       additional_features     accuracy      0.770572
7   Costa_rica            dummy_encoding     accuracy      0.778243
8    Occupancy        occupancy_baseline     accuracy      0.989767
```

Extract date features

- Let's first convert date to `datetime` type

```
occupancy_data['date'] = pd.to_datetime(occupancy_data['date'], format = '%d/%m/%y %H:%M')
```

- Let's extract the date features

```
occupancy_data['day'] = occupancy_data['date'].dt.day
occupancy_data['month'] = occupancy_data['date'].dt.month
occupancy_data['year'] = occupancy_data['date'].dt.year
occupancy_data['hour'] = occupancy_data['date'].dt.hour
occupancy_data['minute'] = occupancy_data['date'].dt.minute
```

Frequency table of date features

- Check the frequency table for year, month and day

```
occupancy_data.year.value_counts()
```

```
2015    8143
Name: year, dtype: int64
```

```
occupancy_data.month.value_counts()
```

```
7     1440
6     1440
9     1440
5     1440
8     1440
10     574
4      369
Name: month, dtype: int64
```

```
occupancy_data.day.value_counts()
```

```
2    8143
Name: day, dtype: int64
```

- `Year` and `day` have only one value so they do not give us much information
- We can add `month` and `hour` as our variables

Add date features to model data

```
dummy_columns = ['hour', 'month']
for i in range(0, len(dummy_columns)):
    colname = "df_" + str(i)
    occupancy_data[dummy_columns[i]] = pd.Categorical(occupancy_data[dummy_columns[i]])
    occupancy_data[dummy_columns[i]] = occupancy_data[dummy_columns[i]].cat.codes
    colname = pd.get_dummies(occupancy_data[dummy_columns[i]], prefix = (str(dummy_columns[i])),
drop_first = True)
    occupancy_data = pd.concat([occupancy_data, colname], axis = 1)
```

- Remove unwanted variables from model data

```
occupancy_X = occupancy_data.drop(['date', 'minute', 'day', 'year', 'hour', 'month'], axis = 1)
```

View the training data

```
occupancy_X.head()
```

```
   Temperature  Humidity  Light     CO2  ...  month_3  month_4  month_5  month_6
0        23.18   27.2720  426.0  721.25  ...        0        0        0        0
1        23.15   27.2675  429.5  714.00  ...        0        0        0        0
2        23.15   27.2450  426.0  713.50  ...        0        0        0        0
3        23.15   27.2000  426.0  708.25  ...        0        0        0        0
4        23.10   27.2000  426.0  704.50  ...        0        0        0        0

[5 rows x 35 columns]
```

Model the data and find accuracy

```
accuracy10 = logistic_model(occupancy_X, occupancy_y)
print(accuracy10)
```

```
0.9995906672124437
```

```
s = pd.Series(['Occupancy', 'with_date_features', 'accuracy', accuracy10], index = ['dataset_name',
'model_name', 'model_metric', 'metric_value'])
performance_df = performance_df.append(s, ignore_index = True)
performance_df
```

	dataset_name	model_name	model_metric	metric_value
0	Costa_rica	baseline_model	accuracy	0.762901
1	Costa_rica	na_imputation_regression	accuracy	0.648187
2	Costa_rica	na_imputation_median	accuracy	0.632845
3	Costa_rica	binning	accuracy	0.767085
4	Costa_rica	categorical_summary	accuracy	0.769177
5	Costa_rica	numerical_summary	accuracy	0.774407
6	Costa_rica	additional_features	accuracy	0.770572
7	Costa_rica	dummy_encoding	accuracy	0.778243
8	Occupancy	occupancy_baseline	accuracy	0.989767
9	Occupancy	with_date_features	accuracy	0.999591

- Our accuracy increased after adding date features when compared to the occupancy baseline model

Spatial features

- Spatial features are location data
- In some data, we will find location-related features such as
  - Latitude
  - Longitude
  - City
  - State
  - Country
  - Zip
- Here is the link to a dataset containing location data variables: ***https://www.kaggle.com/datafiniti/pizza-restaurants-and-the-pizza-they-sell***

- Find the snapshot of the data here

categories	city	country	keys	latitude	longitude
Pizza Place 14% Restaurant 13% Other (559) 73%	Philadelphia 3% New York 3% Other (671) 95%		us/wv/charleston... 2% us/ct/eastgranby... 2% Other (987) 97%	18.4 – 64.9	-158 – -66
Pizza Place	Bend	US	us/or/bend/cascadevillagemallacrossfromtarget/-2134715703	44.10266476	-121.3007971
Pizza Place	Bend	US	us/or/bend/cascadevillagemallacrossfromtarget/-2134715703	44.10266476	-121.3007971
American Restaurant,Bar,Bakery	Los Angeles	US	us/brentwood/losangeles/148sbarringtonave/-1516414008	34.06456347	-118.4690173
American Restaurant,Bar,Bakery	Los Angeles	US	us/brentwood/losangeles/148sbarringtonave/-1516414008	34.06456347	-118.4690173
American Restaurant,Bar,Bakery	Los Angeles	US	us/brentwood/losangeles/148sbarringtonave/-1516414008	34.06456347	-118.4690173

Feature engineering of spatial data

- Mostly spatial features can be kept as categories
- We can convert the categorical data to dummy as we usually do
- Sometimes we can keep the most occurring features alone and the least occurring features together as a single category `other` in order to avoid increasing the dimensions in the data
- `Latitude` and `Longitude` columns can be either removed or kept as numerical data
- They can be sometimes mapped to some external map data source and exact location name can be added as a categorical data

Curse of dimensionality

- Until now, we have seen a lot of techniques to add new features so that our model performance can increase
- But we should not forget, increasing dimensions can lead to complex data and there might be a danger of overfitting
- **We should always make the best judgment in choosing the number of features** needed to model our data with good performance
- There is no hard rule for the right number of features, or what good performance is

Principle of parsimony

- The principle of parsimony in machine learning simply states that we should **explain the most with the least**
- In other words, use the least number of features to explain most of the underlying signal of the data
- If we have 4 features that give us 83% accuracy and adding additional 20 features improves the accuracy by 1%, we can let go of the additional 20 features and have only 4 features
- Our aim should always be to select the optimal number of features needed to get a model with high performance
- **But how do we find the optimal number of features?**

Feature selection techniques

- We already saw a few techniques for feature selection
  - Principle component analysis to choose features with the most variance
  - Feature importance graphs in ensemble methods to choose the top features which model the data better
  - Removing features which have 0 coefficients in lasso regression

Summary

- Today, we saw different techniques of feature engineering to improve our model performance for
  - Numerical data
  - Categorical data
  - Temporal data
  - Spatial data
- We also saw that we need to choose optimal features to find an optimal accuracy
- Always remember that feature engineering is an art and mostly a trial and error approach
- It requires good domain knowledge of the data, so make sure to feel comfortable with your data!

Knowledge check 4

Exercise 4

Module completion checklist

Objective	Complete
Summarize feature engineering and its usefulness	✔
Explain the feature engineering of numerical variables	✔
Implement feature engineering of the numerical variable on Costa Rican dataset	✔
Derive additional features on Costa Rican dataset	✔
Explain the feature engineering of categorical variables	✔
Implement feature engineering of the categorical variable on Costa Rican dataset	✔
Handle temporal and spatial predictors in the data	✔
Explain the principle of parsimony and feature selection	✔

Workshop!

- Workshops are to be completed in the afternoon either with a dataset for a capstone project or with another dataset of your choosing
- Make sure to annotate and comment your code so that it is easy for others to understand what you are doing
- This is an exploratory exercise to get you comfortable with the content we discussed today
- Today you will:
  - Use your own dataset and understand the data first
  - Use different feature engineering techniques and add additional features
  - Use appropriate feature selection method and choose the best model with optimal feature set

This completes our module

Congratulations!