# MSDS 532
# Data Science Programming with Python

## Week 7:
## Regular Expressions, Data Science Project Walk Through

Dr. Eve Thullen
University of Cumberlands

# Overview

**Session 1: Regular Expressions**

- **re** module
- compare string functions
- isolate substrings using patterns

**Session 2: Data Science Project**

- analysis process step: planning, preparing, and analyzing
- with encoding, classification, regression

# Session 1:

## Regular Expressions

UNIVERSITY OF THE CUMBERLANDS

# Regular Expressions

- known as regex, regexp
- concise, flexible means of **pattern-matching strings**
- powerful, yet cryptic, a language of it's own

# Common Symbology in Regex

| | |
|---|---|
| `^` | • beginning of string |
| `$` | • end of string |
| `.` | • any character |
| `\s` | • any whitespace |
| `\S` | • any non-whitespace |
| `*` | • repeats any character zero or more times |

UNIVERSITY OF THE CUMBERLANDS

# Python Regex

- requires installing and importing **`re`**
- common functions **`re.search()`** and **`re.findall()`**

# Base Python `find` vs `re.search`

```python
words = "Python has numerous built-in
functions for strings. Using regex, a
lot more can be accomplished."

print(words.find("lot")) # returns 67
print(words.find("sh")) # returns 91
print(words.find("h")) # returns 3
```

```python
import re

words = "Python has numerous built-in
functions for strings. Using regex, a lot
more can be accomplished."

print(re.search("lot", words))
# <re.Match object; span=(3, 4), match='h'>
```

**Shown here doing the same task**

# Wild Cards

- dot matches any character
- generally wild cards are combined with symbol for any character

```
from: group 1 participant b: 357
from: participant group a1: 352
from: participants in groups: 325
```

both colons were captured

match start of string

many times

```
^f.*:
```

match any character

# Greedy Matching

- plus infers one or more matches of preceding symbol or character
- both wild & plus are greedy

```
from: group 1 participant b: 357
from: participant group a1: 352
from: participants in groups: 325
```

both colons were captured

match start of string

one or more

```
^f.+:
```

match any character

# Non-Greedy Matching

- capture the content up to the first occurrence
- the question mark tempers search

```
from: group 1 participant b: 357
from: participant group a1: 352
from: participants in groups: 325
```

up to and including one colon were captured

match start of string

one or more but not greedy

`^f.+?:`

match any character

UNIVERSITY OF THE CUMBERLANDS

# Match and Extract Substrings

- to **extract** use `re.findall`
- regex `[0-9]` which captures a digit; **+** infers one or more

```
import re
another = "String 12 with 334 random 325 numbers in8it"
print(re.findall('[0-9]+', another))

# returns ['12', '334', '325', '8']
```

one or more

[0-9]+

any number between 0-9

# Regex Groups

- search after, search before type patterns
- use parentheses to identify groups (the string to extract)

```
import re
email = "fake.email@ucumberlands.edu"
print(re.findall("@([^ ]*)", email))

fake.email@ucumberlands.edu
```

everything after symbol,
until blank space

group to return

@([^ ]*)

look for
the symbol

match non-blank
characters

# Special Characters

- numerous special characters have specific meaning in regex, i.e., **?**, **+** , *****, etc.
- patterns with special characters require escape \

```
import re
moola = "Give me $5.00, now."
print(re.findall('\$[0-9.]+', moola))

# returns [$5.00]
```

escape, $ is in string

one or more

**\$[0-9.]+**

digit or period

# Data Cleaning with Regex

- really **powerful tool** working with string data
- consider misspelled words, inconsistent data entry, and validating cleanliness
- each language has regex, with properties unique to *that* language

# Session 2:

## Data Science Project Walk Through

UNIVERSITY OF THE
CUMBERLANDS

# Start the Script

- create script file named **`car_prices.py`**
- **`Goal: Predict the price of the car`**
- add the leading block comment notes
- add the following import statements

```
import pyreadr as pyr
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer # used with one hot encoding
from sklearn.model_selection import train_test_split
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.pipeline import Pipeline
from sklearn import metrics
from sklearn.inspection import permutation_importance
```

# Objective

- What attributes of an automobile have the **largest influence** in the **price** in Great Britain, listed between 2014 and 2021?
- attributes included: make, year advertised, year made, color, miles, body type, fuel type, transmission type, along with the available seating and number of doors

UNIVERSITY OF THE CUMBERLANDS

# Sample

attributes not necessarily included, but represent the data subset

- includes the quantity of class labels to include, by frequency
- 50 **models**
- 6 **colors**
- 2 **body types**
- only automatic and manual **transmissions**
- only diesel and petrol (unleade **fuel**
- 3 **seats**
- 3 **doors**

UNIVERSITY OF THE CUMBERLANDS

# Read Data

- data provided in Blackboard, adapted from Huang et al. (2021)
- use **`pyr.read_r`** to **import data**

```
carAd_file = pyr.read_r("car_ads.RData")
carAd = carAd_file["carAd"]
carAd = pd.DataFrame(carAd) # so pd functions are colored
carAd.reset_index(drop = True, inplace = True)
```

- take a **look at the data**, the data types, and the type of information in each field

# Collection Continued

- imported data is not sole act of collection
- some cases **more than one import**
- data includes information not used
  - don't clean data you're not going to use
  - use **objectives as an outline** in script
  - **subset data for objective**

# Connect Data to Objective

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 268255 entries, 0 to 268254
Data columns (total 16 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Maker          268255 non-null  object
 1   Genmodel       268255 non-null  object
 2   Genmodel_ID    268255 non-null  object
 3   Adv_ID         268255 non-null  object
 4   Adv_year       268255 non-null  int64
 5   Adv_month      268255 non-null  int64
 6   Color          246380 non-null  object
 7   Reg_year       268248 non-null  float64
 8   Bodytype       267301 non-null  object
 9   Runned_Miles   268248 non-null  float32
 10  Engin_size     266191 non-null  object
 11  Gearbox        268088 non-null  object
 12  Fuel_type      267846 non-null  object
 13  Price          268255 non-null  object
 14  Seat_num       261781 non-null  float64
 15  Door_num       263702 non-null  float64
dtypes: float32(1), float64(3), int64(2), object(10)
memory usage: 31.7+ MB
None
```

- vehicle models as **`Genmodel`**
- colors as **`Color`**
- body types as **`Bodytype`**
- transmission types as **`Gearbox`**
- fuel types as **`Fuel_type`**
- seats as **`Seat_num`**
- doors as **`Door_num`**

UNIVERSITY OF THE CUMBERLANDS

# Collect Frequency Recipe

for each column requiring mutation

- collect **frequency by class**
- collect list of **classes to retain**
- compare with frequencies to inspect expected
- **filter data** by retained classes
- collect frequency by class
- **compare before and after** frequencies to inspect
- comment out print calls as you go

collect all lists of retained classes **before mutation**

UNIVERSITY OF THE CUMBERLANDS

# Retained Classes Lists

commented previous inspect calls

outlined objectives

```
# 50 models
# print(carAd["Genmodel"].value_counts(dropna = False))
# captures 0-49 to model classes
mods = carAd["Genmodel"].value_counts().index.tolist()[:50]
# print(mods) # inspect

# 6 colors
print(carAd["Color"].value_counts(dropna = False))
# captures 0-5 top color classes
cols = carAd["Color"].value_counts().index.tolist()[:6]
print(cols) # inspect

# 2 body types
# only automatic and manual transmissions
# only diesel and petrol (unleaded) fuel
# 3 seats
# 3 doors
```

fill in the outline as you go

UNIVERSITY OF THE CUMBERLANDS

# Mutate Data

```
# 50 models using mods list
# print(carAd["Genmodel"].value_counts(dropna = False))
carAd_r1 = carAd[carAd["Genmodel"].isin(mods)]
# print(carAd_r1["Genmodel"].value_counts(dropna = False))

# 6 colors using cols list
print(carAd_r1["Color"].value_counts(dropna = False))
carAd_r2 = carAd_r1[carAd_r1["Color"].isin(cols)]
print(carAd_r2["Color"].value_counts(dropna = False))
```

- make remaining lists for objective **on your own**
- next, apply the lists

DataFrame **name changes**

- compare before and after
- ability to return to earlier version*

# Stage 2: Clean,Remove Fields

- fields not used in analysis
- fields with **only unique values** (or nearly all unique)
- inspect each candidate column
- use **`pandas drop`** method
- inspect expected

```
data_revised = data.drop(columns = "column name")
```

# Stage 2: Explore, Inspection and Clean

- go through **each column**
  - insufficient quantity per class label?
  - any values that are impossible? i.e., negative mileage, engine size 1000L, color is transparent, price in the billions
  - any values that seem unlikely?
  - data types correct?
- **after changing anything, recheck everything**

# Stage2: Revisiting Data Types

- within **`pandas`** is **`category`**-type
  - this type is almost **non-existent** outside of **`pandas`**
  - use **`object`**-type if using other modules
- numbers representing categories are **not numbers** statistically

# Ready For Stage 3: Analyze

- if data is ready, there are **11 columns & 56,388 rows**
- years fields are set to **integer-type**
- mileage max value is over 600K—is this feasible? How old is it? Is this an error?
- price data is now **numeric**; max price is over 200K GBP; is this feasible? What type of vehicle is it? How old is it?
- seats and doors are set to **object-type**

# Planning the model

- using regression & classification with extremely randomized trees algorithm (algorithm by Geurts et al., 2006)
  - outliers aren't a problem; NA is; multicollinearity *can* be
  - no formal statistical assumptions
  - **weaknesses**: impurity-based importance bias
  - **mitigation**: permutation-based importance & imp comparison
- evaluate model fit, validity, reliability with **training and testing**

UNIVERSITY OF THE CUMBERLANDS

# Planning: Recipe for Prep & Analysis

Project Stage:

1. Identify fields requiring **encoding**
2. instantiate encoder & model
3. fit encoder
4. establish pipeline
5. split data for training & testing
6. **train model** & evaluate training performance
7. test model & evaluate testing performance
8. **compare training & testing** for model
9. if model adequate, **evaluate importance** from model (impurity)
10. evaluate permutation importance (Find the important features)
11. **compare importance features**

regression only
for right now

UNIVERSITY OF THE CUMBERLANDS

# Recipe for Prep & Analysis

- identify fields requiring **encoding**

```
objs = carAd_r8.select_dtypes(include = np.object_).columns.tolist()
```

- **instantiate** encoder & model

```
ohe = OneHotEncoder()
cTrans = make_column_transformer((ohe, objs), remainder = "passthrough")
# instantiate model; random state is seeding; any integer is accepted
# max features to none means use all features
# verbose: 0 = show me nothing; 1 = give me time hacks; 2 = tell me everything
etr = ExtraTreesRegressor(random_state = 75, max_features = None, verbose = 1)
```

# Recipe Continued

- fit encoder

```
cTrans.fit(carAd_r8)  # exculde price column
# fit transformer to ALL data (so test is transformed the same way as training)
```

- establish pipeline

```
pipe = Pipeline(steps = [('ctrf', cTrans), ('model', etr)])
```

- split data for training & testing

```
x,testX,y,testY = train_test_split(carAd_r8.iloc[:,carAd_r8.columns!=[Price"],
        carAd_r8["Price"],
        test_size = .3,
        stratify = carAd_r8["Maker"], # chosen due to the number of classes
        random_state = 1)
```

# Assessing Performance

- train model & evaluate **training performance**

```
pipe.fit(x, y)  # training <- this may take a few minutes!
print("R\N{SUPERSCRIPT TWO} =", pipe.score(x, y))
# R² = 0.9980646255482711
trPr = pipe.predict(x)
print("RMSE =",metrics.mean_squared_error(y,trPr,squared=False))
# RMSE = 457.11440511573124
```

- test model & evaluate **testing performance**

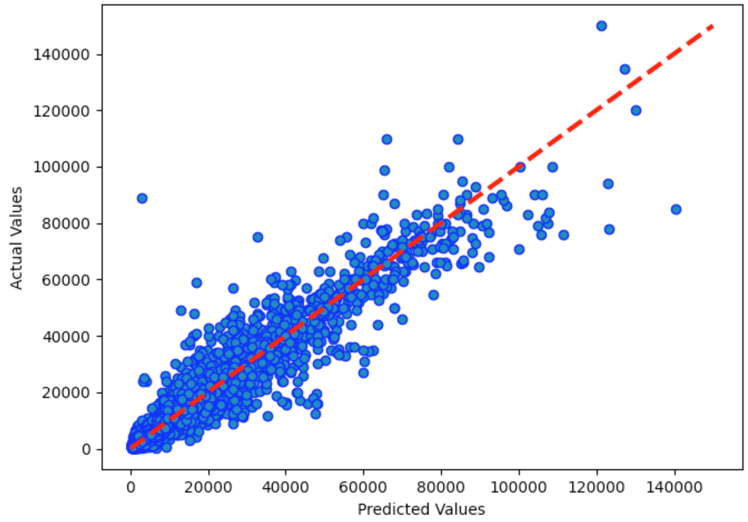
```
pred = pipe.predict(testX)
print("R\N{SUPERSCRIPT TWO} =",metrics.explained_variance_score(testY,pred))
# R² = 0.9152397309162292
print("RMSE =",metrics.mean_squared_error(testY,pred,squared = False))
# RMSE = 3151.612939664625
```

- compare training & testing for model**

# Visualize Testing Performance

test model & evaluate **testing performance** visualization

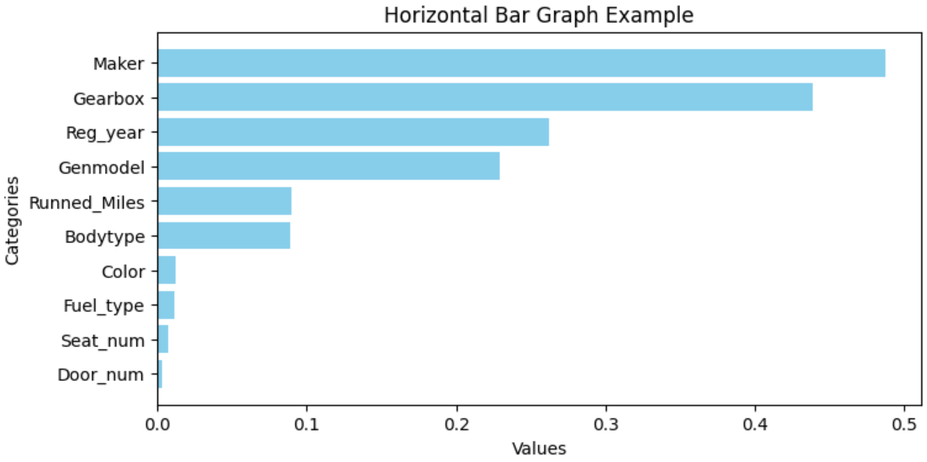
```
# visualize testing performance
fig, ax = plt.subplots()
ax.scatter(pred, testY, edgecolors = (0,0,1))
ax.plot([testY.min(), testY.max()],
        [testY.min(), testY.max()],
         'r--', lw = 3)
ax.set_xlabel('Predicted Values')
ax.set_ylabel('Actual Values')
plt.show(block = True)
```




UNIVERSITY OF THE CUMBERLANDS

# Importance Columns

- from sklearn.inspection import permutation_importance
  - # a list of all column names of independent variables
  - indeps = carAd9.iloc[:, carAd8.columns != "Price"].columns.tolist()
  - perm_imps = permutation_importance(pipe, trainx, trainy, n_repeats = 5, random_state = 1)
  - # get features order for permutation
  - perms_df = pd.DataFrame({"Features": indeps, "Permutation": perm_imps.importances_mean}).sort_values(by = "Permutation", ascending = True)




```
        Features  Permutation
0          Maker     0.487104
6        Gearbox     0.438498
3       Reg_year     0.262281
1       Genmodel     0.229046
5   Runned_Miles     0.090056
4       Bodytype     0.089094
2          Color     0.012131
7      Fuel_type     0.011631
8       Seat_num     0.007285
9       Door_num     0.003252
Features        object
Permutation    float64
```

# Recipe for Importance

- evaluate **importance** from model (impurity)
  - **collect values**

```
imps = pipe.named_steps["model"].feature_importances_
```

  - collect labels (columns and encoded labels)
  - print out table

# Importance Features

```
print(pipe.named_steps["ctrf"].named_transformers_["onehotencoder"].categories_)
     # a list of lists of encoded columns names
one_hot_cat = pipe.named_steps["ctrf"].named_transformers_["onehotencoder"].categories_
     # a list of all column names of independent variables
indeps = carAd_r8.iloc[:, carAd_r8.columns != "Price"].columns.tolist()

# list to house the labels that were used in the model
features = []
for each in indeps:              # create label list for importance
    if each in objs:
        spot = objs.index(each)          # get position index
        # for each one hot label, prefix column name and underscore to label
        one_hot_mess = [each + "_" + str(what) for what in one_hot_cat[spot]]
                                         # str() because of seats and doors
        features.extend(one_hot_mess)    # extend a list with another iterable
    else:
        features.append(each)            # append a single item to the list
print(features)            # feature labels sent to model
```

**collect labels** (columns and encoded labels)

UNIVERSITY OF THE CUMBERLANDS

# Importance Feature Table

```
influencers =
pd.DataFrame({"Features":
features, "Importance":
imps}).sort_values(by =
"Importance", ascending =
False)

print(influencers)
```

| Features | Importance |
|---|---|
| Genmodel_GLE Class | 0.002116 |
| Color_White | 0.002175 |
| Genmodel_Panamera | 0.002197 |
| Bodytype_Combi Van | 0.002291 |
| Genmodel_Amg Gt | 0.002512 |
| Genmodel_S Class | 0.002581 |
| Gearbox_Semi-Automatic | 0.002615 |
| Color_Black | 0.002633 |
| Color_Grey | 0.002636 |
| Maker_Rolls-Royce | 0.002692 |
| Fuel_type_Hybrid Petrol/Electric | 0.002828 |
| Door_num | 0.002830 |
| Genmodel_Phantom | 0.002848 |
| Genmodel_F430 | 0.002902 |
| Genmodel_488 | 0.002987 |
| Maker_BMW | 0.003152 |
| Genmodel_NSX | 0.003201 |
| Bodytype_Limousine | 0.003447 |
| Bodytype_Pickup | 0.003907 |
| Genmodel_R8 | 0.004360 |
| Genmodel_SLS | 0.005450 |
| Maker_Aston Martin | 0.005810 |
| Genmodel_Range Rover | 0.005838 |
| Seat_num | 0.006362 |

# Collect Permutation

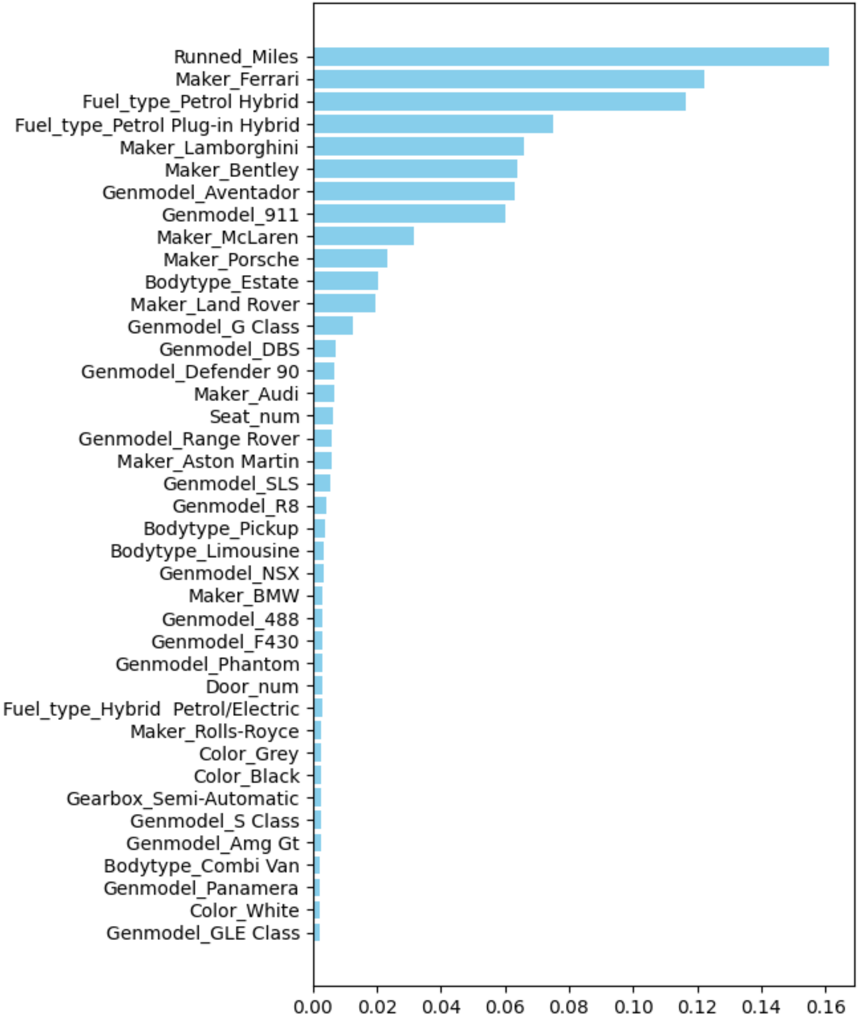
```
# due to weakness in ETR
perm_imps = permutation_importance(pipe, x, y, n_repeats = 5, random_state = 1)
# create an array to use for sorting
perm_sorts = perm_imps.importances_mean.argsort()
print(perm_imps.importances[perm_sorts])
# create an offset index for plotting
indices = np.arange(0, len(imps)) + .5
# get features order for permutation
perms_df = pd.DataFrame({"Features": indeps,
    "Permutation": perm_imps.importances_mean}).sort_values(by = "Permutation",
                                                          ascending = False)
```

evaluate **permutation importance**

# Importance Visualization

```
# visualize differences between importance measures
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (12, 8))
# horizontal bar graph of original importance values
ax1.barh(indices, imps[np.argsort(imps)], height = .7)
ax1.set_yticks(indices)
ax1.set_yticklabels(influencers["Features"].tolist().reverse())
ax1.set_ylim((0, len(imps)))
# box plot showing permutation importance across n_repeats
ax2.boxplot(perm_imps.importances[perm_sorts].T,
            vert = False, labels = perms_df["Features"].tolist(),
            patch_artist = True,
            boxprops = {'color': "black", 'edgecolor': "black"})
fig.tight_layout()
plt.show(block = True) # show me!
```

**comparing importance**





# Importance Visualized

# Classification

- same construct as regression; except outcome variable is categorical (although annotated *object*-type)
- do **not** encode outcome variable

```
from sklearn.ensemble import ExtraTreesClassifer
```

- encoding and pipelining works the **same**, as does importance
- **performance evaluation is a lot different**

UNIVERSITY OF THE CUMBERLANDS

# Classification Performance

- use **confusion matrix** and **classification report**
- **avoid using accuracy** to assess performance*
- no information rate (NIR)—accuracy by guessing
  - if two class labels exist, evenly dispersed in data
  - NIR is 50% ← guess and get that
  - if your model is no better than guessing…

# Calling Performance

```
# create list of possible classes
#    these are the unique values in the dependent variable
dep_labels = df["outcomeVariable"].value_counts().index.tolist()
# print(dep_labels) # inspect
print("------ Training Performance ------")
pred_fit = pipe.predict(trainX)
print(metrics.confusion_matrix(ytrain, pred_fit, labels = dep_labels))
print(metrics.classification_report(ytrain, pred_fit, labels = dep_labels))

print("------ Testing Performance ------")
pred_fit = pipe.predict(testX)
print(metrics.confusion_matrix(ytest, pred_etc, labels = dep_labels))
print(metrics.classification_report(ytest, pred_etc, labels = dep_labels))
```

# Understanding Performance

- classification and regression performance addressed in Guttag (2021)
- setting **doors as the outcome variable** & assessing influence should suggest that price is the biggest influence** try it out!
- this analysis assesses relationships—**not cause!!**

# Q&A: Week 7 Assignments

- **Practical Connection: Course Reflection**

  Provide a reflection of at least 500 words (or 2 pages double spaced) of how the knowledge, skills, or theories of this course have been applied or could be applied in a practical manner to your current work environment. If you are not currently working, share times when you have or could observe these theories and knowledge that could be applied to an employment opportunity in your field of study.

- **Problem 6 Set: Playing Word Game**
  - Submit your .py files
  - Submit your final testing results