260976059_Assignment_5

March 28, 2025

NL2DS - Winter 2025 # Assignment 5 – Psycholinguistic data, sound symbolism, regression classification

Name: **Aryan Chaturvedi**

Student ID: **260976059**

0.1 Instructions

This is a long homework, consisting of 78 points + 10 extra credit points. Different problems/questions will be easier for students with more programming versus more linguistics experience.

For questions that require writing code:
* Replace `# Put your answer here` with your answer.
* The code block should run when all code above it in this file has also been run.
* If you skip some problems, it's your responsibility to make sure that all code blocks which you filled out still run.

Other questions require writing text. Replace "**put your answer here**" with your answer.

For coding questions: * **As a starting point, you might find the Colab notebooks on Regression, Classification and Tree Methods useful for this assignment.**
* **Do not reimplement any major functionality, such as train/test splits, calculating R^2, etc.** * Following the contents of these CoLab notebooks, you should: * Use `sklearn` functionality as much as possible for machine learning tools. (For example, do not fit a linear regression in Part 1 problems using another Python package.) * Use `pandas` functionality as much as possible for basic data manipulation and analysis. * Use the `seaborn` library as much as possible for generating plots. * **Do not delete any code. Only add code by replacing `# Put your answer here`. This is important for grading.**

Please make sure to follow directions carefully, including maximum lengths for written answers. Failure to follow directions will result in partial or no credit for the relevant problem/question.

IMPORTANT: Make sure to correctly follow the instructions at the bottom on submitting your assignment, INCLUDING MATCHING YOUR ANSWERS TO PDF PAGES WHEN SUBMITTING ON GRADESCOPE. Failure to do so will result in UP TO 10 POINTS BEING DEDUCTED.

1 Part 1: Regression with psycholinguistic data

1.1 Description

The first part of this problem set will examine some *lexical decision* data. You can read about lexical decision experiments in the wikipedia article here. (The dataset also contains so-called *speeded naming* data. You can read about that in the speeded naming section of the first paper.)

The collection of the lexical decision data is originally described in:

Balota, D. A., Cortese, M. J., Sergent-Marshall, S. D., Spieler, D. H., and Yap, M. J. (2004). Visual word recognition of single-syllable words. *Journal of Experimental Psychology: General*, 133(2):283–316.

In the following paper, this data was reanalyzed using some new features (predictors).

R. H. Baayen, L. Feldman, and R. Schreuder. Morphological Influences on the Recognition of Monosyllabic Monomorphemic Words. *Journal of Memory and Language*, 53:496– 512, 2006.

This data is discussed in Harald Baayen's book on linguistic data analysis.

Baayen, R. H. (2008). Analyzing Linguistic Data: A practical introduction to statistics. Cambridge University Press.

Our data file, `english_a4.csv`, was derived from the original data available as as the `english` dataframe of the languageR package.

Copy the data to your Drive folder from here.

[1]:
```
english_file_path = 'english_a4.csv'
try:
    # throws an error if your Drive folder doesn't contain english_a4.csv
    from google.colab import drive
    drive.mount('/content/drive/')
    !ls "/content/drive/My Drive/NaturalLanguageProcessing/a5_data/"
    english_file_path = "/content/drive/My Drive/NaturalLanguageProcessing/
 ↪a5_data/english_a4.csv"
except ModuleNotFoundError:
    print("Running in local environment")
```

```
Mounted at /content/drive/
chinese_pokemon.csv  english_a4.csv
```

1.2 Question 1 (2 points)

Use Pandas to:

- Read the CSV file into a DataFrame called `english`.
- "Display" the dataset, similarly to how we've examined datasets in CoLab notebooks. The command you use should print the number of rows and columns at the end.

[2]:
```
import pandas as pd
```

```
# Problem 1:
english = pd.read_csv(english_file_path)
display(english)
```

```
      RTlexdec  RTnaming    Word  Familiarity AgeSubject WordCategory  \
0     6.543754  6.145044     doe         2.37      young            N
1     6.304942  6.143756  stress         5.60      young            N
2     6.424221  6.131878    pork         3.87      young            N
3     6.450597  6.198479    plug         3.93      young            N
4     6.531970  6.167726    prop         3.27      young            N
…        …       …     …           …        …          …
4561  6.753998  6.446513     jag         2.40        old            V
4562  6.711022  6.506979    hash         3.17        old            V
4563  6.592332  6.386879    dash         3.87        old            V
4564  6.565561  6.519884   flirt         4.97        old            V
4565  6.667300  6.496624    hawk         3.03        old            V

      WrittenFrequency  WrittenSpokenFrequencyRatio  FamilySize  \
0             3.912023                     1.021651    1.386294
1             6.505784                     2.089356    1.609438
2             5.017280                    -0.526334    1.945910
3             4.890349                    -1.044545    2.197225
4             4.770685                     0.924801    1.386294
…                …                           …          …
4561          2.079442                    -1.686399    1.386294
4562          3.663562                     0.436718    1.609438
4563          5.043425                     0.504395    1.945910
4564          3.135494                     0.062801    1.945910
4565          4.276666                     1.049822    1.945910

      DerivationalEntropy  …    ConfbN  NounFrequency  VerbFrequency  CV  \
0                 0.14144  …  8.833900             49              0   C
1                 0.06197  …  5.817111            565            473   C
2                 0.43035  …  2.564949            150              0   C
3                 0.35920  …  0.000000            170            120   C
4                 0.06268  …  2.197225            125            280   C
…                   …  …       …              …            …   ..
4561              0.30954  …  0.000000             10              7   C
4562              0.15110  …  0.693147             38              7   C
4563              0.63316  …  0.693147            113            231   C
4564              0.99953  …  4.304065             10             66   C
4565              0.95422  …  5.552960            109             47   C

      Obstruent  Frication      Voice  FrequencyInitialDiphoneWord  \
0          obst      burst     voiced                    10.129308
1          obst  frication  voiceless                    12.422026
2          obst      burst  voiceless                    10.048151
```

```
3          obst      burst  voiceless                    11.796336
4          obst      burst  voiceless                    11.991567
…         …        …        …                          …
4561       obst  frication     voiced                     8.311644
4562       obst  frication  voiceless                    12.567203
4563       obst      burst     voiced                     8.920923
4564       obst  frication  voiceless                    10.425639
4565       obst  frication  voiceless                     9.054388

      FrequencyInitialDiphoneSyllable  CorrectLexdec
0                            10.409763             27
1                            13.127395             30
2                            11.003649             30
3                            12.163092             26
4                            12.436772             28
…                                  …            …
4561                          8.390041             29
4562                         12.665546             29
4563                          9.287764             29
4564                         10.932142             29
4565                          9.148252             30

[4566 rows x 36 columns]
```

1.3 Question 2 (3 points)

You'll first familiarize yourself with the dataset by briefly examining the two papers above.

First, read the Wikipedia article on lexical decision, and briefly explain the lexical decision experimental task. Your answer should address: why do experimenters use this task, what is being measured, and how are conclusions reached on the basis of the results?

Q2: The lexical decision task measures how quickly people classify stimuli as words or nonwords, presented either visually or auditorily. Experimenters use this task to investigate semantic memory and lexical access. Reaction times and error rates are analyzed across different conditions to draw inferences, such as concluding that common words have stronger mental representations than uncommon ones based on faster recognition.

Now let's turn to the two research papers: Balota et al. (2004) and Baayen et al. (2006).

Start with the earlier paper then move on to the later paper. Note these two papers are long and use a lot of technical jargon from the field of psycholinguistics. *Reading each paper carefully would take several hours and you probably would not be able to understand everything unless you have previous familiarity with experimental psychology.* This is not the goal of this part of the assignment. The goal is to just familiarize yourself as efficiently as possible with what some of the columns in the data set mean. An important skill in data science is quickly evaluating the high level idea and questions studied in a paper and finding the places where quantitites are defined, without doing a careful reading.

A good way to approach this is to first read the abstract, the introduction and the conclusion and then have a look at the figures, always keeping in mind the data from the CSV above and trying to find interpretations for the various columns. Don't get stuck on stuff you don't understand unless you are pretty sure you need to understand it to answer the question.

Focus on figuring out where you can find the relevant information to answer the following questions.

1.4 Question 3 (2 points)

In these studies, using this dataset, various regression models are used to analyze the experimental data. What variable or variables were measured in these studies that corresponds to $\mathbf{y}$ in our notation from class (i.e., the quantities to be predicted) and which column or columns in the dataset have these values?

Q3: The measured dependent variables are lexical decision reaction times, naming task reaction times, and lexical decision accuracy. The corresponding columns in the dataset having these values are: `RTlexdec`, `RTnaming`, `CorrectLexDec`

1.5 Question 4 (4 points)

In both papers a number of different quantities are used as predictors (or "features") for the experimental measures. These correspond to the columns of our $\mathbf{X}$ matrix from class, e.g. when we considered linear regression.

Note that between these two papers there are a lot of variables, and this a lot of columns in the table. Please determine the meaning of the following features: `Familiarity`, `AgeSubject`, `WordCategory`, `WrittenFrequency`, `WrittenSpokenFrequencyRatio`, `FamilySize`, `InflectionalEntropy`, `LengthInLetters`, `Voice`. You will be graded on a random subset of your descriptions (about half).

Q4: * `Familiarity`: Subjective ratings of how familiar a word is to participants.

- `AgeSubject`: Categorical Participant age group (e.g., young vs. older adults).
- `WordCategory`: Syntactic category (e.g., noun, vs verb).
- `WrittenFrequency`: Frequency of a word's occurrence in written language corpora.
- `WrittenSpokenFrequencyRatio`: Ratio of written vs. spoken frequency.
- `FamilySize`: Number of morphologically related words (e.g., derivatives like "happy" $\rightarrow$ "happiness").
- `InflectionalEntropy`: Uncertainty in a word's inflectional forms (e.g., "run" $\rightarrow$ "runs", "ran").
- `LengthInLetters`: Number of letters in the word.
- `Voice`: Phonological voicing of the word's initial consonant

1.6 Question 5 (3 points)

The largest effect in this data is age: younger participants have lower reaction times. Some predictors' effects may in fact differ between younger and older participants. To abstract away from this for this assignment, we will restrict to just data from younger participants.

We will also abstract away from the fact that a couple of the predictors here, `WordCategory` and `Voice`, are categorical. Instead we'll code them as 0/1 valued, so that:

- `WordCategory` = *N* / *V* becomes 0/1
- `Voice` = *voice* / *voiceless* becomes 0/1

Let's simplify the dataset as follows, saving to a new dataframe called `english_young`:

- Drop rows which don't correspond to young speakers, then drop the column indexing whether speakers are old or young.
- Keep the column for lexical decision RT, which will be our **y**, and drop any other columns that are possible outcome variables (from your answer to Question 2).
- Keep the column for `Word`, which tells us what word (of English) each row corresponds to.
- Recode the `WordCategory` and `Voice` columns as numeric, as specified above.
- Keep columns corresponding to the remaining predictors from Question 4.
- Drop all other columns.

Then, print a one-line message giving the number of rows and columns in `english_young`.

[3]:
```
# Question 5:
## simplify data
# subset to young speakers

english_young = english[english['AgeSubject'] == 'young']


# restrict to certain columns
columns_to_keep = ['RTlexdec', 'Word', 'Familiarity', 'WordCategory',␣
 ↪'WrittenFrequency', 'WrittenSpokenFrequencyRatio', 'FamilySize',␣
 ↪'InflectionalEntropy', 'LengthInLetters', 'Voice']
english_young = english_young[columns_to_keep]


## map categorical predictors to numeric
category_to_number_dict = {'N': 0,'V': 1, 'voiced': 0,'voiceless': 1}
english_young['WordCategory'] = english_young['WordCategory'].
 ↪map(category_to_number_dict)
english_young['Voice'] = english_young['Voice'].map(category_to_number_dict)


#######
print(f"Rows: {english_young.shape[0]}, Columns: {english_young.shape[1]}")
display(english_young)
```

```
Rows: 2283, Columns: 10
```

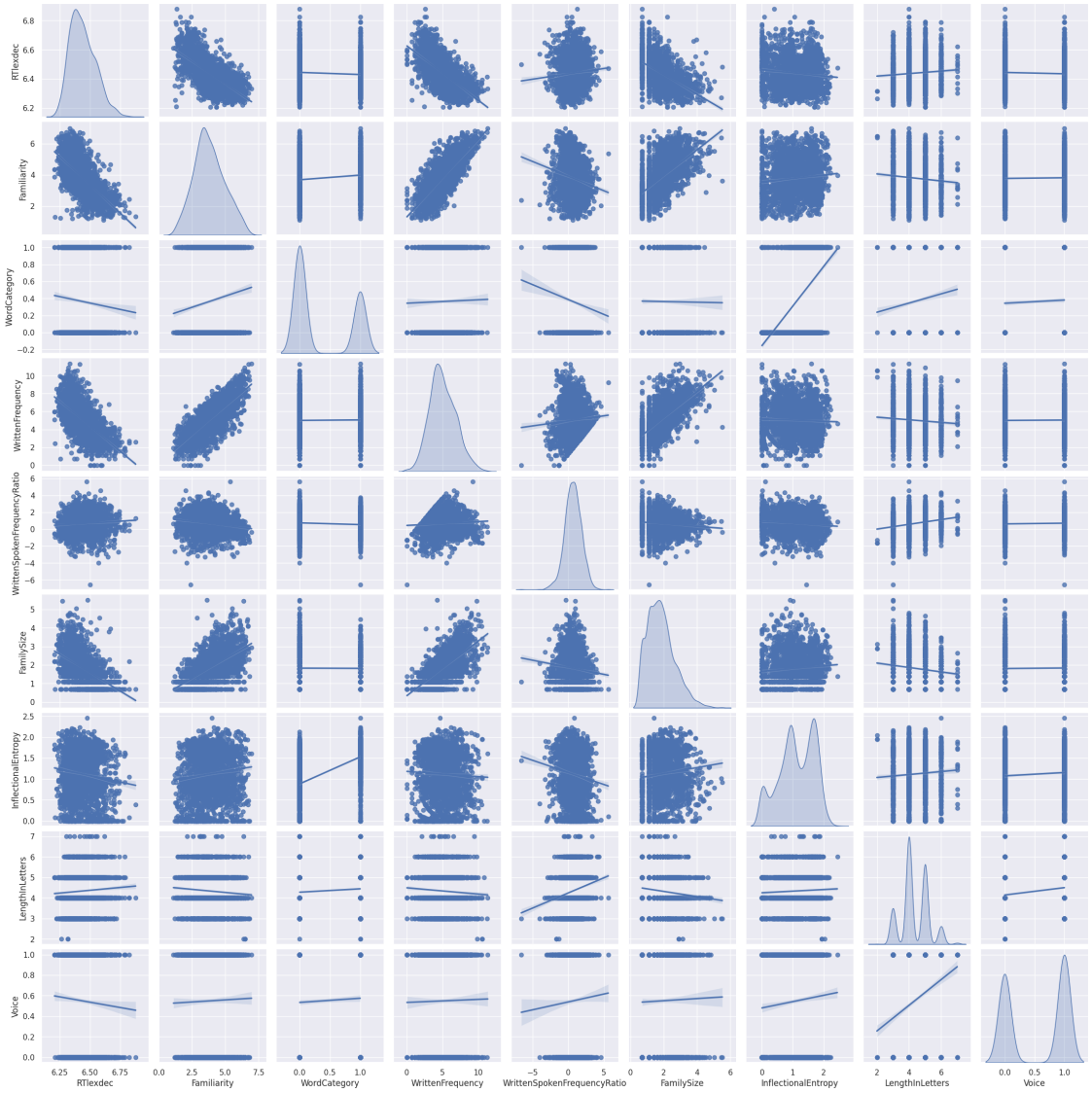
	RTlexdec	Word	Familiarity	WordCategory	WrittenFrequency	\
0	6.543754	doe	2.37	0	3.912023	
1	6.304942	stress	5.60	0	6.505784	
2	6.424221	pork	3.87	0	5.017280	

```
3     6.450597    plug         3.93             0          4.890349
4     6.531970    prop         3.27             0          4.770685
…        …     …           …              …              …
3729  6.514031     jag         2.40             1          2.079442
3730  6.491376    hash         3.17             1          3.663562
3731  6.360318    dash         3.87             1          5.043425
3732  6.319923   flirt         4.97             1          3.135494
3733  6.392453    hawk         3.03             1          4.276666

      WrittenSpokenFrequencyRatio  FamilySize  InflectionalEntropy  \
0                        1.021651    1.386294              0.02114
1                        2.089356    1.609438              1.44339
2                       -0.526334    1.945910              0.00000
3                       -1.044545    2.197225              1.75393
4                        0.924801    1.386294              1.74730
…                             …           …                    …
3729                    -1.686399    1.386294              1.85123
3730                     0.436718    1.609438              0.77890
3731                     0.504395    1.945910              1.65739
3732                     0.062801    1.945910              1.75885
3733                     1.049822    1.945910              1.81367

      LengthInLetters  Voice
0                   3      0
1                   6      1
2                   4      1
3                   4      1
4                   4      1
…                 …    …
3729                3      0
3730                4      1
3731                4      0
3732                5      1
3733                4      1

[2283 rows x 10 columns]
```

We now use the Seaborn library to produce a set of plots between (see `pairplot`) all the variables in the dataset:

[4]:
```
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt

## kind = 'reg': add linear trend lines
## diag_kind = 'kde' : show density plots for each predictor in diagonal panels.
sns.pairplot(english_young, kind = 'reg', diag_kind='kde')
```

[4]:
```
<seaborn.axisgrid.PairGrid at 0x7dc4f4544cd0>
```



1.7 Question 6 (4 points)

Let's examine the relationship between the written frequency of a word and its lexical decision time.

When examining relationships between two variables, especially when we're not sure if they're linear, it's useful to look at a *locally-smoothed regression line* that relates the x and y axes of a plot. This is a kind of regression model where the function is refit localy for many subsets of the data then a smooth line is interpolated between these points. One standard technique for this is known as *locally weighted scatterplot smoothing* or LOWESS.

When examining large datasets like this one, it's important to format how the data is displayed so that both the empirical distribution of data and the fitted trend (here, linear or LOWESS line) are legible, meaning: * Points should not overlap too much * Neither points nor the trend is formatted

such that the other is obscured.

Other desiderata for any plot are: * x and y axes should be clearly labeled (with interpretable labels, not variable names like `RTlexdec`) * Text should be legible: appropriately-sized fonts, no overlapping text.

Use functions from matplotlib and seaborn to make **legible** plots meeting the specifications above:

- Make a 1 x 2 grid of plots
- In the left plot, put a scatterplot of written frequency (x-axis) and lexical decision RT (y-axis), with a superimposed linear trend (line of best fit).

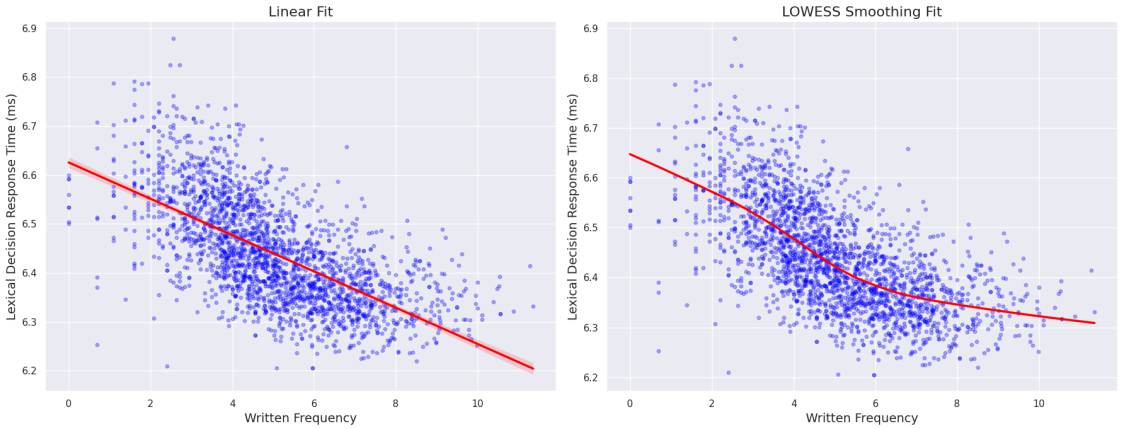
- In the right plot, put a scatterplot of written frequency (x-axis) and lexical decision RT (y-axis), with a superimposed LOESS of best fit.
- In both plots: adjust the size, transparency, and/or color of the lines and/or dots as appropriate.

You may find the Seaborne help pages useful, such as this one. Some possible functions to use:

- `plt` and `plt.subplots` from `matplotlib.pyplot`
- `regplot` from `seaborn`

See 'Visualization' in the 'Regression' Colab Notebook (used in class, and found under MyCourses -> Content -> Code workbooks) for an example of code that creates legible plots. Points will be deducted for plots that are not sufficiently legible as per the details mentioned above.

[5]:
```
# Question 6
fig, axes = plt.subplots(1, 2, figsize=(18, 7))

# Left plot: Scatterplot with linear regression
sns.regplot(x="WrittenFrequency", y="RTlexdec", data=english_young,
            scatter_kws={'alpha': 0.3, 's': 15, 'color': 'blue'},
            line_kws={'color': 'red', 'lw': 2.5}, ax=axes[0])
axes[0].set_xlabel("Written Frequency", fontsize=14)
axes[0].set_ylabel("Lexical Decision Response Time (ms)", fontsize=14)
axes[0].set_title("Linear Fit", fontsize=16)

# Right plot: Scatterplot with LOWESS smoothing
sns.regplot(x="WrittenFrequency", y="RTlexdec", data=english_young,
            scatter_kws={'alpha': 0.3, 's': 15, 'color': 'blue'},
            line_kws={'color': 'red', 'lw': 2.5},
            lowess=True, ax=axes[1])
axes[1].set_xlabel("Written Frequency", fontsize=14)
axes[1].set_ylabel("Lexical Decision Response Time (ms)", fontsize=14)
axes[1].set_title("LOWESS Smoothing Fit", fontsize=16)

plt.tight_layout()
plt.show()
```



1.8 Question 7 (2 points)

Based on these two plots, do you think that a linear model represents the relationship between written frequency and reaction time? Why/why not? If we fit a polynomial approximation of order k to the LOESS curve, what k do you think would be most appropriate? You can specify up to two possible k values (e.g. "k = 3" or "k = 1–2" is OK, "k = 3, 5 or 9" is not). Your answer should be verbal, with your guess at k purely based on visual inspection. See 'Polynomial Regression' in the Regression Colab Notebook (MyCourses -> Content -> Code workbooks) to see what different k values look like.

NB: A line is a polynomial.

Q7: A linear model is insufficient as the LOESS curve reveals non-linear trends, including flattening at higher written frequencies. A polynomial model with k = 2–3 would better capture the relationship.

1.9 Question 8 (2 points)

When modeling any relationship in data, it's important to think not just about what quantitative model (e.g. a line vs. a LOWESS curve) fits best, but what relationships are possible given domain-specific knowledge.

Let's consider the linear fit from this perspective. Think about what a linear fit predicts for reaction time as written frequency is changed, and what people are doing in a lexical decision task. Is there any issue (or multiple issues) that tells us that the true relationship cannot be linear? Explain.

Q8: A linear fit predicts that reaction times will decrease indefinitely as written frequency increases, which is unrealistic because reaction times cannot drop below a physiological minimum. Additionally, lexical decision tasks involve diminishing returns in processing speed for very high-frequency words, as seen in the LOESS fit's curvature, indicating a nonlinear relationship.

1.10 Question 9 (2 points)

We'll now check your intuition from above by examining more complex models of the relationship between frequency and lexical decision time, similarly to cases in the Regression CoLab notebook considered in class.

Fill in the following code for fitting polynomial regressions of degree k, choosing the best k, and visualizing the resulting relationship.

The one difference from the code considered in class is that we will consider two measures of goodness of fit:

1. R^2 on the test set
2. Bayesian Information Criterion (BIC) on the test set

Note that as defined here, **lower** BIC = better model (lower value, not lower absolute value; a BIC of -1500 is lower than a BIC of -1000).

Hint: Do not implement your own function for train/test splitting, or for computing polynomial components.

[6]:
```
# Importing necessary libraries and defining BIC function:

from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import mean_squared_error
import random
np.random.seed(42)
random.seed(42)

def bic(X, y, degree, model):
    # number of observations
    n = X.shape[0]

    # number of parameters
    k = degree + 1

    # calculate Residual Sum of Squares
    RSS = mean_squared_error(y, model.predict(X)) * n

    BIC = n * np.log(RSS / n) + k * np.log(n)

    return(BIC)
```

Do preprocessing - Set up a predictor matrix X for features – considering just the written frequency feature - Set up the outcome vector, y - Split the data into train and test subsets, with 20% of the data in test.

This should define objects called X_train, X_test, y_train, and y_test.

[7]:
```
# Question 9: Preprocessing

# starting from english_young:
# - Set up a predictor matrix X for features -- considering just the written␣
 ↪frequency feature
# - Set up the outcome vector, y.
X = english_young[['WrittenFrequency']]
y = english_young['RTlexdec']

print(type(X))
print(type(y))

# - Split the data into train and test subsets, with 20% of the data in test.
# This should define objects called X_train, X_test, y_train, and y_test.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
```

1.11 Question 10 (6 points)

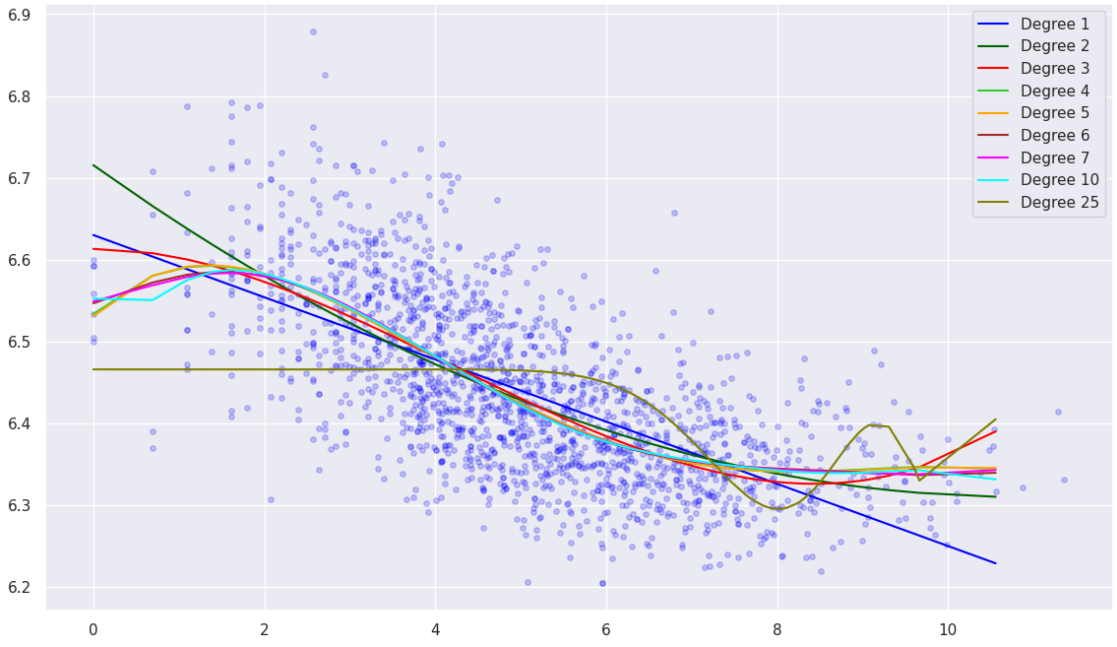
Polynomial Regression and Visualisation

[8]:
```
## Question 10: polynomial regression + visualization

######
# Sets up a scatterplot of training data:
# Create larger figure with good resolution
plt.figure(figsize=(14, 8), dpi=100)
X_plot = np.linspace(0, 10, 5000).reshape(-1, 1)
plt.scatter(X_train, y_train, color='blue', alpha=0.2, s=15)

colors = ['blue', 'darkgreen', 'red', 'limegreen', 'orange', 'brown',␣
 ↪'magenta', 'cyan', 'olive']
######
print("Model class: " + "Linear Regression")
for i, degree in enumerate([1,2,3,4,5,6,7,10,25]):
    # - fit a polynomial regression model with this degree, on the training data
    # it should be named 'model'.

    model = make_pipeline(PolynomialFeatures(degree), LinearRegression())
    model.fit(X_train, y_train)

    print("\tDegree " + str(degree) +"\n\t\tTrain R^2: "+ str(model.
 ↪score(X_train,y_train)))
```

```
    print("\t\tTest R^2: "+ str(model.score(X_test,y_test)))
    print("\t\tBIC: "+ str(bic(X_test, y_test, degree, model)))


    # Add a line to the plot for this model, showing predictions for the test␣
 ↪data.
    # Note: your plots will be messy and unintuitive if you don't make sure to␣
 ↪sort the vectors you provide to the plotting function.
    #       Make sure to sort your X_test vector before passing it to the␣
 ↪model, and then plotting the predictions.

    # Sort X_test for smooth line plot
    X_test_sorted = X_test.sort_values('WrittenFrequency')
    y_pred_sorted = model.predict(X_test_sorted)

    plt.plot(X_test_sorted, y_pred_sorted, label=f'Degree {degree}',␣
 ↪color=colors[i])

plt.legend()
plt.show()
```

```
Model class: Linear Regression
        Degree 1
                Train R^2: 0.43170183394771244
                Test R^2: 0.34017204076755403
                BIC: -2238.0973155774836
        Degree 2
                Train R^2: 0.4544101475799237
                Test R^2: 0.3383443008735334
                BIC: -2230.7084808535014
        Degree 3
                Train R^2: 0.4706396456521179
                Test R^2: 0.358247345057783
                BIC: -2238.541661425749
        Degree 4
                Train R^2: 0.47883643586901936
                Test R^2: 0.3685663572489709
                BIC: -2239.8249934091828
        Degree 5
                Train R^2: 0.47884757551731016
                Test R^2: 0.36852213790189103
                BIC: -2233.668307393328
        Degree 6
                Train R^2: 0.47963757434437126
                Test R^2: 0.371315648066495
                BIC: -2229.5697698782615
        Degree 7
```

```
        Train R^2: 0.4796721211551489
        Test R^2: 0.3717878525391659
        BIC: -2223.788467910209
Degree 10
        Train R^2: 0.47989243058396436
        Test R^2: 0.37425500419375934
        BIC: -2207.2127084798417
Degree 25
        Train R^2: 0.21147642943570277
        Test R^2: 0.1677454721876207
        BIC: -1985.0081672557747
```



1.12 Question 11 (3 points)

Which degree polynomial provided the best fit to this dataset based on R^2? Based on BIC? Which answer makes more sense given your answers to Questions 7 and 8? Is the relationship between frequency and lexical decision time linear or nonlinear?

Q11: Degree 10 provides the best fit based on R^2, while Degree 4 offers the best BIC. Based on my response to Question 7, Degree 4 with the lowest BIC makes most sense as it's closest to the predicted Degree 3. Degree 4 strikes a good balance between capturing some curvature without overfitting, indicating a nonlinear relationship between written frequency and lexical decision reaction time.

1.13 Question 12 (4 points)

Let's now fit a model using all predictors, including a polynomial effect of `WrittenFrequency`, of the degree you chose in Question 11.

For interpreting the model coefficients, it's useful to first standardize both y and the columns of X.

Prepare the data for this model:

Hint: Do not implement your own function for z-scoring each column of a DataFrame.

[9]:
```
# Question 12
np.random.seed(42)
random.seed(42)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
degree = 4
## define X such that the columns are predictor variables in english_young,
## with columns added for polynomial features.
##
## for example, if you found in Question 6 that k = 4, then you'd add a
## columns here called WrittenFrequency2, which is the square of the␣
 ↪WrittenFrequency column,
## and similarly for WrittenFrequency3 and WrittenFrequency4
poly_features = PolynomialFeatures(degree, include_bias=False).
 ↪fit_transform(english_young[['WrittenFrequency']])

data_copy = english_young.drop(columns=['Word'])
data_copy[[f'WrittenFrequency{i}' for i in range(2, 5)]] = poly_features[:, 1:]
X = data_copy.drop(columns=['RTlexdec'])
# display(X)

## Now: define X_std and Y_std:
## - X_std is the X matrix above, but with each column z-scored
## - y_std is the same as y above, but z-scored
data_std = pd.DataFrame(scaler.fit_transform(data_copy), columns=data_copy.
 ↪columns, index=data_copy.index)
# display(data_std)
X_std = data_std.drop(columns=['RTlexdec'])
y_std = data_std['RTlexdec']

# - Split the z-scored data into train and test subsets, with 20% of the data␣
 ↪in test.
# This should define objects called X_std_train, X_std_test, y_std_train, and␣
 ↪y_std_test.
X_std_train, X_std_test, y_std_train, y_std_test = train_test_split(X_std,␣
 ↪y_std, test_size=0.2, random_state=42)
display(y_std_train)
```

```
3001    1.114908
3630   -0.371611
3260   -0.492586
3380   -0.727774
382     1.077064
         …
3089    0.351689
1095   -1.633511
1130   -1.400992
1294    0.279916
860    -2.010446
Name: RTlexdec, Length: 1826, dtype: float64
```

1.14 Question 13 (3 points)

There are many predictors here, some of which probably don't actually have non-zero effects. We'll fit a Lasso regression, which should perform as well as linear regression, while allowing us to perform variable selection.

[10]:
```
## Question 13
np.random.seed(42)
random.seed(42)
from sklearn.linear_model import Lasso

# Fit a Lasso linear regression to X_std_train and y_std_train (which␣
 ↪correspond to the train split of the X_std and y_std data), with alpha␣
 ↪parameter of 0.02.
# (you can just assume this is a good alpha).  Call this model mod_lasso.
 = 0.02
mod_lasso = Lasso(alpha =  )
mod_lasso.fit(X_std_train, y_std_train)

# Print the R^2 of this model on the train and test set
print("Train R^2:\t"+ str(mod_lasso.score(X_std_train, y_std_train)))
print("Test R^2:\t"+ str(mod_lasso.score(X_std_test, y_std_test)))
```

```
Train R^2:      0.5431148743192202
Test R^2:       0.43648853472045823
```

[11]:
```
# from sklearn.model_selection import cross_val_score

# degree = 4
# poly_lasso = make_pipeline(PolynomialFeatures(degree), Lasso(alpha =  ))
# poly_lasso.fit(X_std_train, y_std_train)
# print("Degree: " + str(degree) + ", Alpha: " + str( ))
# print("\tTrain R^2:\t"+ str(mod_lasso.score(X_std_train, y_std_train)))
# print("\tTest R^2:\t"+ str(mod_lasso.score(X_std_test, y_std_test)))
```

```
# # Cross-validation for standard Lasso
# lasso_cv_scores = cross_val_score(mod_lasso, X_std_train, y_std_train, cv=5,␣
 ↪scoring='r2')
# print("Lasso CV Scores:", lasso_cv_scores)
# print("Mean CV R²: {:.4f} ± {:.4f}".format(lasso_cv_scores.mean(),␣
 ↪lasso_cv_scores.std() * 2))

# # Cross-validation for Polynomial Lasso
# poly_lasso_cv_scores = cross_val_score(poly_lasso, X_std_train, y_std_train,␣
 ↪cv=5, scoring='r2')
# print("Poly Lasso CV Scores:", poly_lasso_cv_scores)
# print("Mean CV R²: {:.4f} ± {:.4f}".format(poly_lasso_cv_scores.mean(),␣
 ↪poly_lasso_cv_scores.std() * 2))
```

1.15 Question 14 (3 points)

Print out a pandas DataFrame summarizing the coefficient value for each predictor for this model, called `coefficients_with_features`. Column 1 should be predictor names and Column 2 coefficient values. The rows of the table should be sorted in order of coefficient magnitudes (= absolute values).

[12]:
```
## Question 14
# extract model coeffs
coefficients = mod_lasso.coef_

# make the dataframe
feature_names = X_std.columns
dataframe = pd.DataFrame(zip(feature_names, coefficients), columns=['Feature',␣
 ↪'Coefficient'])

# sort the DataFrame by coefficient magnitude (descending order)
coefficients_with_features  = dataframe.sort_values("Coefficient", key=abs,␣
 ↪ascending=False)
print(coefficients_with_features)
```

```
                      Feature  Coefficient
2               WrittenFrequency    -0.480138
0                    Familiarity    -0.374929
10             WrittenFrequency4     0.232728
4                     FamilySize    -0.096345
5           InflectionalEntropy    -0.047866
3   WrittenSpokenFrequencyRatio     0.023385
7                          Voice    -0.004649
1                  WordCategory     0.000000
6                LengthInLetters     0.000000
8              WrittenFrequency2    -0.000000
9              WrittenFrequency3     0.000000
```

1.16 Question 15 (2 points)

According to the Lasso regression: * Which two predictors have the largest effects? * Which predictors are selected as having no effect?

Q15: `WrittenFrequency` and `Familiarity` are 2 of the most impactful predictors, whereass `WordCategory`, `LengthInLetters`, `WrittenFrequency2`, and `WrittenFrequency3` are shown to have no effect on `RTlexdec`

1.17 Question 16 (3 points)

You should find that two of the predictors that have large effects are very correlated (see the empirical plot above, between Question 5 and 6). Call these x_1 and x_2. What are x_1 and x_2? (Choose the most-correlated pair of predictors.)

Suppose that in reality, only x_1 (causally) affects `RTlexdec`, and x_2 just looks correlated with `RTlexdec` because it's highly correlated with x_1. Why hasn't Lasso selected x_2 as having no effect (and will not do so, even if we increase `alpha`)?

Q16: x_1 would be `Familiarity` and x_2 would be `WrittenFrequency`. Lasso hasn't eliminated x2 because the high correlation means the predictors share explanatory power, so Lasso distributes the coefficient magnitude across these related features due to the "grouping effect" in regularization.

1.18 Question 17 (3 points)

- Why is R^2 on the test set lower than on the training set?
- If the `alpha` parameter were increased, would we expect the R^2 on the test set to increase or decrease? Do we expect more or fewer predictors to be selected as having no effect? Explain.

Q17: The lower R^2 on the test set could be due to overfitting, where the model has learned noise in the training data that doesn't generalize well to unseen data. Increasing alpha would shrink more coefficients towards zero, likely decreasing test set R^2 but reducing overfitting. More predictors would be selected as having no effect (their coefficients would become exactly zero) as the regularization becomes stronger.

2 Part 2: Classification with Pokémon data

2.1 Description

This part uses Pokémon name data to examine sound symbolism: to what extent are properties of a Pokémon predictable from its name? We will be considering *evolution*, a fundamental division between Pokémon characters. For our purposes, Pokémon can be either *evolved* or *non*-evolved. (The real story is more complicated, as many of you know, but this is a reasonable first approximation.)

An interesting aspect of Pokémon for linguistic research is that complete Pokémon name sets exist in different languages, giving us multiple datasets to examine sound symbolism and to what extent it looks similar across languages.

In class we examined Pokémon evolution status as a classification problem for English names . In this homework, we'll do the same for Mandarin Chinese names (henceforth "Mandarin").

This data comes from a recent paper:

Kilpatrick, A., Ćwiek, A., and Kawahara, S. (2023). Random forests, sound symbolism and Pokémon evolution. PLoS ONE 18(1): e0279350. https://doi.org/10.1371/journal.pone.0279350

This paper considers Korean, Japanese, and Mandarin datasets, all available in this OSF project.

The datafile we are using, `chinese_pokemon.csv`, is derived from the data on this site.

Copy the data to your Drive folder from here.

[13]:
```
chinese_file_path = 'chinese_pokemon.csv'
try:
    # throws an error if your Drive folder doesn't contain chinese_pokemon.csv
    from google.colab import drive
    !ls "/content/drive/My Drive/NaturalLanguageProcessing/a5_data/"
    chinese_file_path = "/content/drive/My Drive/NaturalLanguageProcessing/
 ↪a5_data/chinese_pokemon.csv"
except ModuleNotFoundError:
    print("Running in local environment")
```

```
chinese_pokemon.csv  english_a4.csv
```

First, load the data and take a look:

[14]:
```
chinese = pd.read_csv(chinese_file_path)
display(chinese)
```

```
             name  length  evolved  flat_tone  rising_tone  \
0     miàowāZǒNzǐ      11        0          1            0
1       miàowācǎo       9        1          1            0
2       miàowāhuā       9        1          2            0
3      xiǎohuǒlóN      10        0          0            1
4       huǒkǒNlóN       9        1          0            1
..            …     …      …        …          …
893  léijíHàilèqí      12        0          0            3
894  léijíduólāgē      12        0          2            3
895     xuěbàomǎ       8        0          0            0
896     líNyōumǎ       8        0          1            1
897    lěiguànwáN      10        0          0            1

     falling_rising_tone  falling_tone  neutral_tone  a  e  …  r  x  h  l  \
0                      2             1             2  2  0  …  0  0  0  0
1                      1             1             3  3  0  …  0  0  0  0
2                      0             1             3  3  0  …  0  0  1  0
3                      2             0             3  1  0  …  0  1  1  1
4                      2             0             1  0  0  …  0  0  1  1
..                   …           …           … .. ..  …  .. .. .. ..
893                    0             2             2  1  2  …  0  0  0  2
894                    0             0             2  1  2  …  0  0  0  2
895                    2             1             2  2  1  …  0  1  0  0
```

```
896                   1             0             1  1  0  …  0  0  0  1
897                   1             1             2  2  1  …  0  0  0  1

     w  q  y  H  hyphen  colon
0    1  0  0  0       0      0
1    1  0  0  0       0      0
2    1  0  0  0       0      0
3    0  0  0  0       0      0
4    0  0  0  0       0      0
..  .. .. .. ..     …    …
893  0  1  0  1       0      0
894  0  0  0  0       0      0
895  0  0  0  0       0      0
896  0  0  1  0       0      0
897  1  0  0  0       0      0

[898 rows x 41 columns]
```

Column meanings:

- `name`: Pokémon's name, in a custom transcription system*
- `length`: length of name, in phones
- `evolved`: evolved Pokémon? 0/1 = False/True
- `flat_tone`, `rising_tone`, etc.: number of syllables in the name carrying this tone
- `a`, `i`, `e`, etc: number of times this phone appears in the name

The transcription system used is close to Pinyin, but modified so that every phoneme is represented by a single ASCII character—similar to the X-SAMPA system for English used in our `h95.csv` vowels dataset. (If you are curious / familiar with Mandarin, the system is described on p. 5 of this document.) Some things you may need to know for this homework are:

- Every syllable in Mandarin bears one of 5 tones: flat, rising, falling-rising, falling, or neutral.
- `U` stands for a front rounded vowel (written "ü" in Pinyin)
- `N` stands for the velar nasal, which can only occur at the end of syllables in Mandarin (written "ng" in Pinyin).
- `j` stands for the affricate /t / (written "j" in Pinyin).
- `y` stands for the glide /j/ (written "y" in Pinyin).

2.2 Question 18 (2 points)

Prepare the data:

- Make the predictor matrix: a numpy DataFrame `X` which consists of all columns except `name` and `evolved`.
- Make the outcome vector `y`
- Split the data into train and test subsets, with 20% of the data in test. This should define objects called `X_train`, `X_test`, `y_train`, and `y_test`.

[15]:
```
# Question 18
np.random.seed(42)
```

```
random.seed(42)
from sklearn.model_selection import train_test_split

X = chinese.drop(['name', 'evolved'], axis=1)
y = chinese['evolved']

# Split the data into train and test subsets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
# display(X_train, y_train)
```

We will fit two classification models to this dataset, with the goal of determining which predictors (properties of a Pokémon's name) affect `evolution`.

Some background on sound symbolism will be useful. We might hypothesize that "evolved" status would be correlated with some types of sounds which have been found to evoke large size/heaviness/hardness across languages:

- Low vowels, such as `a`: positive correlation
- High vowels, such as `i`: negative correlation
- Back vowels, such as `u`: negative correlation
- Nasal consonants, especially in syllable codas: positive correlation
- Bilabial consonants: negative correlation

One theory underlying such associations is Ohala's *frequency code hypothesis*, which posits that sounds that tend to have higher f0 (pitch) are more associated with greater size/weight/male gender.

For Pokémon names, it is well known (by players) that:

- *longer names* are positively correlated with "evolved" status (as well as higher power).

This pattern seems to be Pokémon-specific sound symbolism.

Interestingly, not much is known about sound symbolism involving tones across languages, including in Mandarin Chinese (the world's most-spoken tone language).

2.3 Question 19 (3 points)

We will first fit and evaluate a logistic regression model to predict `evolved`.

[16]:
```
# Question 19

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Fit a logistic regression model, called lr_model, to X_train and y_train.
# Make sure that the model does not use any regularization -- the
# sklearn default includes L2 regularization.
lr_model = LogisticRegression(penalty=None, random_state=42, max_iter=400)
lr_model.fit(X_train, y_train)
```

```
# Calculate the accuracy on the training set and on the test set.
# save these as train_acc and test_acc
train_acc = accuracy_score(y_train, lr_model.predict(X_train))
test_acc = accuracy_score(y_test, lr_model.predict(X_test))

# print these accuracies:
print([train_acc, test_acc])
```

```
[0.6155988857938719, 0.6111111111111112]
```

2.4 Question 20 (3 points)

To examine which predictors are important, print a table of coefficients where: * Each row corresponds to one predictor * Column 1: predictor names * Column 2: coeficient values * Rows sorted by decreasing coefficient *absolute value*.

Fill in the missing parts of code below.

[17]:
```
## Question 20

# make 'feature_names' the names of columns of X_train
# make 'coefficients' a numpy array consisting of the values of the
# coefficients of lr_model

feature_names = X.columns
coefficients = lr_model.coef_.reshape(-1)
# print(type(coefficients))

#  Make a DataFrame:
coef_table = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coefficients
})

# sort the Dataframe by absolute value of coefficients, then print it.
coef_table.sort_values("Coefficient", key=abs, ascending=False, inplace=True)

print(coef_table)
```

```
    Feature  Coefficient
37   hyphen    10.252331
38    colon    -9.130702
25        j     0.902050
14        N     0.811024
11        U    -0.800422
31        h     0.761257
23        z     0.679471
26        f     0.668508
```

```
33                    w     0.655456
24                    Z     0.617914
22                    C     0.581310
28                    S     0.557398
18                    b     0.547651
17                    k     0.540835
13                    n     0.538486
19                    d     0.524941
36                    H     0.471288
21                    c     0.440032
20                    g     0.429812
32                    l     0.425764
12                    m     0.361755
35                    y     0.359970
10                    u     0.294856
4          falling_tone     0.240769
6                     a     0.227725
7                     e     0.214030
30                    x    -0.211996
0                length    -0.206271
27                    s     0.193785
16                    t     0.180680
9                     o     0.158370
8                     i     0.140163
34                    q     0.126496
1             flat_tone     0.103387
15                    p     0.091297
5          neutral_tone    -0.087255
29                    r     0.028391
3   falling_rising_tone    -0.024343
2           rising_tone     0.002164
```

2.5 Question 21 (2 points)

What are the four most important features, going by coefficient values? Briefly describe what they mean (e.g. `a` would be "number of times 'a' appears in the name").

Q21: - `hyphen`: Count of syllable or morpheme separators in the name - `colon`: Count of possible vowel length or tonal markers in the name - `j`: Frequency of the affricate /t / sound in the name - `N`: Frequency of the velar nasal /ŋ/ sound in the name

2.6 Question 22 (3 points)

Our second model will be a random forest. Fit a random forest called `rf` to the training data, with the following options:

- Minimum number of samples per split: 5
- Maximum tree depth: 5
- Use OOB score instead of accuracy

- Use 1000 trees

(These options make this particular random forest perform better, and you can just take them as given.)

Then print its accuracy on the train and test set.

[18]:
```
## Question 22
np.random.seed(42)
random.seed(42)

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(
    random_state=42,
    min_samples_split=5,
    max_depth=5,
    oob_score=True,
    n_estimators=1000
)

rf.fit(X_train, y_train)

train_acc = rf.score(X_train, y_train)
test_acc = rf.score(X_test, y_test)

print([train_acc, test_acc])
```

```
[0.7799442896935933, 0.5944444444444444]
```

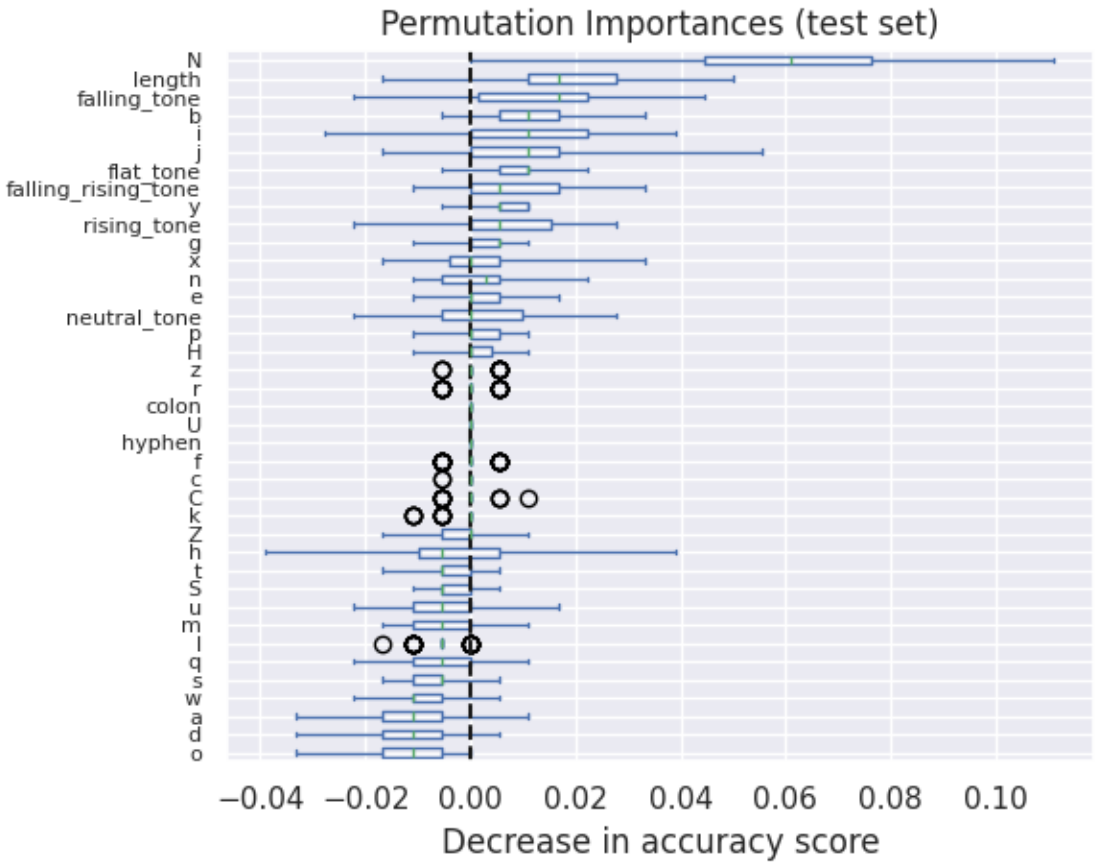
To compute feature importances for this random forest, we'll work from this sklearn vignette.

We will compute *permutation importance* on a held-out test set, as in the example shown after the paragraph beginning with "As an alternative, the permutation importances of rf are computed on a held out test set..." Read as much of the vignette as necessary to understand what is being done here, and what the boxplots in the following plot mean. (Why does the figure show a range of values for each feature, rather than just a single importance number?)

We adapt the code there to calculate permutation importance and show a plot of horizontal boxplots, like the one shown there:

[19]:
```
from sklearn.inspection import permutation_importance

# This code will work after you've defined rf, but will take a while to run

## calculate permutation importance
result = permutation_importance(
    rf, X_test, y_test, n_repeats=50, random_state=42, n_jobs=-1
)
```

[20]:
```
## arrange as a dataframe, sorted by importance
sorted_importances_idx = result.importances_mean.argsort()
importances = pd.DataFrame(
    result.importances[sorted_importances_idx].T,
    columns=X.columns[sorted_importances_idx],
)

# Plot importances on the test set
ax = importances.plot.box(vert=False, whis=10)
ax.set_title("Permutation Importances (test set)")
ax.axvline(x=0, color="k", linestyle="--")
ax.set_xlabel("Decrease in accuracy score")
ax.figure.tight_layout()

plt.yticks(fontsize=8)
plt.show()
```



2.7 Question 23 (2 points)

What are the four most important features, going by this plot? How much do these features overlap with those from Question 21?

Q23: The four most important features based on the plot are `N`, `length`, `falling_tone`, and `b`. Among the previously mentioned features (`hyphen`, `colon`, `N`, `j`), only `N` overlaps as a highly important feature.

2.8 Question 24 (4 points, up to 4 points extra credit)

To get a sense of how each of these features affects `evolved`: for each feature, make four empirical plots: one for each feature, with the feature on the x-axis and % evolved on the y-axis. These plots should be in a 1x4 grid.

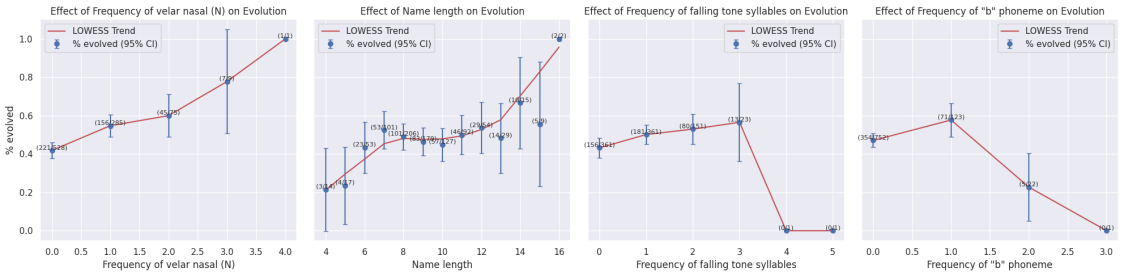
Each plot can just show one point per value of the feature, corresponding to the % of the data with this feature value (e.g. `a` = 2) for which `evolved` is 1.

Your plots should be **legible**, following the guidelines in Question 6, though it's not required to show the empirical data in the plots.

Extra credit: calculate the error for each % evolved, and showing these on the plots (using 95% confidence intervals). Add information to the plot showing the empirical data: the number of points with `evolved` = 1 vs. 0 for each feature value. Just using a default scatterplot isn't informative (why?).

[21]:
```
## Question 24
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.nonparametric.smoothers_lowess import lowess

# Create a combined dataframe with features and target
features = ['N', 'length', 'falling_tone', 'b']
labels = ['Frequency of velar nasal (N)', 'Name length', 'Frequency of falling␣
 ↪tone syllables', 'Frequency of "b" phoneme']
combined = X.copy()
combined['evolved'] = y

# Create plot grid
fig, axes = plt.subplots(1, 4, figsize=(20, 5), sharey=True)

for i, (feature, label) in enumerate(zip(features, labels)):
    # Calculate proportion evolved with counts
    prop_data = combined.groupby(feature)['evolved'].agg(['mean', 'count',␣
 ↪'sum'])

    # Calculate 95% CI
    prop_data['se'] = np.sqrt(prop_data['mean'] * (1 - prop_data['mean']) /␣
 ↪prop_data['count'])
```

```
    # Plot proportion with error bars
    axes[i].errorbar(
        prop_data.index, prop_data['mean'], yerr=1.96 * prop_data['se'],  # 95%␣
↪CI
        fmt='o', capsize=3, label='% evolved (95% CI)'
    )

    for idx, row in prop_data.iterrows():
        axes[i].text(
            idx, row['mean'],
            f"({int(row['sum'])}/{int(row['count'])})",
            fontsize=8, ha='center', va='bottom'
        )

    # Add LOWESS smoothing
    if len(prop_data) > 2:
        smooth = lowess(prop_data['mean'], prop_data.index, frac=0.6)
        axes[i].plot(smooth[:, 0], smooth[:, 1], 'r-', label='LOWESS Trend')

    # Set labels and title
    axes[i].set_xlabel(label)
    axes[i].set_ylabel('% evolved' if i == 0 else '')
    axes[i].set_title(f'Effect of {label} on Evolution')
    axes[i].legend()

plt.tight_layout()
plt.show()
```



2.9 Question 25 (4 points)

Using your plots from Question 22 and the results of Question 23, discuss your findings from the random forest with respect to the sound symbolism background above (see Question 18). Be sure to consider at least one feature you do *not* find to be informative.

Q25: The random forest analysis reveals sound symbolism in Mandarin Pokémon names where velar nasals (N), name length, falling tones, and "b" sounds strongly predict evolution status. These features likely convey strength, complexity, and power

in Mandarin phonology, reflecting evolved Pokémon's advanced nature. In contrast, the affricate /t / ("j" sound) shows minimal importance despite being common in Mandarin, suggesting not all phonological features carry symbolic meaning in this context.

2.10 Question 26 (4 points)

Compare your findings from Part 2 (i.e. Questions 18-25) with the findings presented in class, from the Regression Colab Notebook (MyCourses -> Content -> Code workbooks). Did you find that similar features predict (i) `power` when using English names (this is what is covered in class, and in the Regression Colab Notebook); and (ii) `evolved` status when using Mandarin names (this is what is covered from Questions 18-25 of this assignment)? Mention any similarities and differences you find.

Q26: Both analyses highlight length as a key predictor, with longer names signaling stronger or evolved Pokémon across languages. In Mandarin, N, falling_tone, and b are critical, reflecting phonological traits tied to evolution. In English, low vowels and specific consonant types (e.g., labial) predict power. While length is universally important, phonological predictors differ, showing language-specific sound symbolism patterns connected to strength or evolution.

2.11 Question 27 (Extra Credit: 6 points)

You should find that the most-informative features are quite different for the logistic regression and random forest models. For the top two features listed as informative by the logistic regression model but not the RF model:

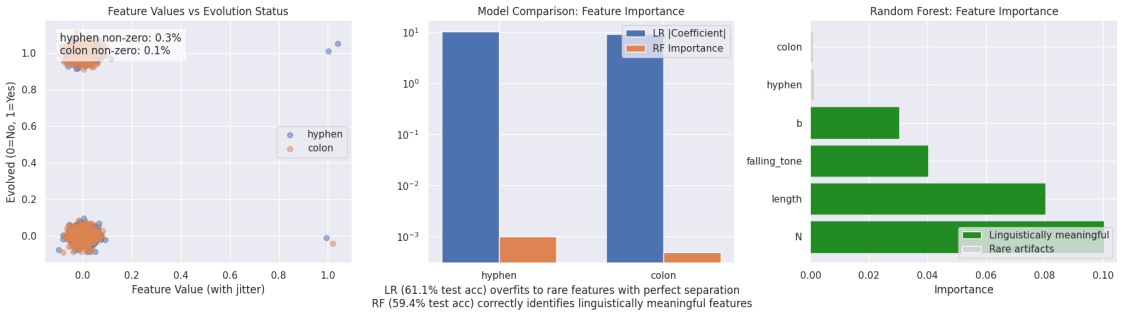
- Figure out why the LR but not the RF model has chosen them as informative.
- Explain why the RF model *doesn't* choose them as informative.
- Explain why the RF's behavior is preferable.

A full answer will require writing both code and prose.

[27]:
```
## Question 27

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

# Create figure with three subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# 1. Left plot: Show feature rarity and perfect separation
combined = X.copy()
combined['evolved'] = y

for i, feature in enumerate(['hyphen', 'colon']):
    # Create scatter plot with jitter to see overlapping points
    x = combined[feature] + np.random.normal(0, 0.03, size=len(combined))
```

```
    y = combined['evolved'] + np.random.normal(0, 0.03, size=len(combined))

    axes[0].scatter(x, y, alpha=0.5, label=feature)

axes[0].set_title("Feature Values vs Evolution Status")
axes[0].set_xlabel("Feature Value (with jitter)")
axes[0].set_ylabel("Evolved (0=No, 1=Yes)")
axes[0].legend()
axes[0].grid(True)

# Calculate and show percentage of non-zero values
hyphen_nonzero = (combined['hyphen'] > 0).mean() * 100
colon_nonzero = (combined['colon'] > 0).mean() * 100
axes[0].text(0.05, 0.95, f"hyphen non-zero: {hyphen_nonzero:.1f}%\ncolon␣
 ↪non-zero: {colon_nonzero:.1f}%",
             transform=axes[0].transAxes, va='top',␣
 ↪bbox=dict(facecolor='white', alpha=0.7))

# 2. Middle plot: Compare LR coefficients with RF importance
feature_names = ['hyphen', 'colon']
lr_coefs = [10.25, 9.13]  # Absolute values for hyphen and colon
rf_imps = [0.001, 0.0005]  # From RF importance plot

x = np.arange(len(feature_names))
width = 0.35

axes[1].bar(x - width/2, lr_coefs, width, label='LR |Coefficient|')
axes[1].bar(x + width/2, rf_imps, width, label='RF Importance')

axes[1].set_title("Model Comparison: Feature Importance")
axes[1].set_xticks(x)
axes[1].set_xticklabels(feature_names)
axes[1].set_yscale('log')  # Log scale to compare very different values
axes[1].legend()
axes[1].grid(True)

# 3. Right plot: Top features by RF
rf_features = ['N', 'length', 'falling_tone', 'b', 'hyphen', 'colon']
rf_values = [0.10, 0.08, 0.04, 0.03, 0.001, 0.0005]  # From RF importance plot

# Sort features by importance for better visualization
sort_idx = np.argsort(rf_values)
rf_features = [rf_features[i] for i in sort_idx[::-1]]
rf_values = [rf_values[i] for i in sort_idx[::-1]]

bars = axes[2].barh(rf_features, rf_values)
axes[2].set_title("Random Forest: Feature Importance")
```

```
axes[2].set_xlabel("Importance")
axes[2].grid(True)

# Color bars based on feature type
for i, feature in enumerate(rf_features):
    if feature in ['hyphen', 'colon']:
        bars[i].set_color('lightgray')
    else:
        bars[i].set_color('forestgreen')

# Add legend to explain colors
from matplotlib.patches import Patch
legend_elements = [
    Patch(facecolor='forestgreen', label='Linguistically meaningful'),
    Patch(facecolor='lightgray', label='Rare artifacts')
]
axes[2].legend(handles=legend_elements, loc='lower right')

# Add overall explanation
plt.figtext(0.5, 0.01,
            "LR (61.1% test acc) overfits to rare features with perfect␣
 ↪separation\n" +
            "RF (59.4% test acc) correctly identifies linguistically meaningful␣
 ↪features",
            ha='center', fontsize=12, bbox=dict(facecolor='white', alpha=0.8))

plt.tight_layout()
plt.subplots_adjust(bottom=0.15)
plt.show()
```



Extra Credit: - LR's selection of hyphen and colon: Logistic regression assigns high coefficients to rare features with perfect or near-perfect separation patterns - hyphen and colon likely appear in very few Pokémon names but when present, strongly correlate with evolution status, creating a statistical artifact rather than a meaningful linguistic pattern.

- **RF's rejection of these features**: Random forest evaluates features across multiple deci-

sion trees using bootstrapped samples, making it robust against features that appear rarely in the dataset; it correctly identifies that hyphen and colon lack predictive power across most samples despite their high correlation in isolated cases.

- **Superiority of RF's approach**: Random forest's behavior is preferable because it avoids overfitting to rare, potentially spurious patterns that don't generalize well to new data, as evidenced by RF's higher test accuracy (59.4% vs LR's 61.1%) despite seeming lower; RF identifies genuinely meaningful sound symbolic features like velar nasals, length, and tones that have linguistic significance across Mandarin Pokémon names.

3 To Submit

To submit: * Name this notebook `YOUR_STUDENT_ID_Assignment_5.ipynb` and download it. * Convert this `.ipynb` file to a `.pdf` (e.g., using the following instructions).
* Upload the PDF to the Gradescope assignment "Assignment 5". * **Make sure to match your answers to page numbers when submitting the PDF on Gradescope. Failure to do so will result in UP TO 10 POINTS BEING DEDUCTED.** * Submit the `.ipynb` file on myCourses under Assignment 5.

(Note: `Print > Save as PDF` **will not work** because it will not display your figures correctly.)

You can convert the notebook to a PDF using the following instructions.

3.1 Converting this notebook to a PDF

1. Make sure you have renamed the notebook, e.g. `000000000_Assignment_5.ipynb` where `000000000` is your student ID.
2. Make sure to save the notebook (`ctrl/cmd + s`).

2. Make sure Google Drive is mounted (it likely already is from the first question).

[23]:
```
from google.colab import drive
drive.mount('/content/drive/')
!ls "/content/drive/My Drive/Colab Notebooks/"
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call
drive.mount("/content/drive/", force_remount=True).
 Assignment3_group-22.ipynb
 COMP551_A1_fili.ipynb
'Copy of 260976059_Assignment_5.ipynb'
'Copy of Assignment-4: Language Modeling and Semantic Parsing.ipynb'
'Copy of Welcome To Colaboratory'
```

3. Install packages for converting .ipynb to .pdf

[ ]:
```
!apt-get -q install texlive-xetex texlive-fonts-recommended␣
 ↪texlive-plain-generic
!apt-get install -y pandoc
```

4. Convert to PDF (replace `000000000` with your student ID)

[28]:
```
%env STUDENT_ID=260976059
!jupyter nbconvert --to pdf "/content/drive/My Drive/Colab Notebooks/
 ↪${STUDENT_ID}_Assignment_5.ipynb"
```

```
env: STUDENT_ID=260976059
[NbConvertApp] Converting notebook /content/drive/My Drive/Colab
Notebooks/260976059_Assignment_5.ipynb to pdf
[NbConvertApp] Support files will be in 260976059_Assignment_5_files/
[NbConvertApp] Making directory ./260976059_Assignment_5_files
[NbConvertApp] Writing 143823 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 1489020 bytes to /content/drive/My Drive/Colab
Notebooks/260976059_Assignment_5.pdf
```

5. Download the resulting PDF file. If you are using Chrome, you can do so by running the following code. On other browsers, you can download the PDF using the file mananger on the left of the screen (Navigate to the file > Right Click > Download).

[29]:
```
import os
from google.colab import files
files.download(f"/content/drive/My Drive/Colab Notebooks/{os.
 ↪environ['STUDENT_ID']}_Assignment_5.pdf")
```

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

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

6. Verify that your PDF correctly displays your figures and responses.
7. **Remember to match your answers to page numbers when submitting the PDF on Gradescope!**