Project: Checkpoint 2

Migraine

Team Members: Suraj Khamkar | Rahul More | Pragathi Pendem | Sravani Pati

GitHub Link: https://github.com/khamkarsuraj/Applied_Data_Science.git

Learnings from Checkpoint 1:

The datasets we have chosen are ERR4796171.fastq and ERR4796172.fastq. To get a greater number of records we have merged dataset in file named as merged_ERR479617a.fastq. This merged file had 314232 records in it before data cleaning consists of duplicate and null values. Secondly, we checked the duplicate values and created another dataset after deleting null values from the same.

The additional data cleaning steps we could have performed are correcting inconsistency, handling outliers, standardizing data and validation data, etc.

Additional data cleaning steps for Checkpoint 2:

[2]:
```
import ast
import statistics
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split
from Bio import SeqIO
import pandas as pd

# Open the FASTQ file
with open('ERR4796171.fastq', 'r') as f:
    # Parse the FASTQ file and create a list of dictionaries with the data
    records = [{'seq_id': record.id,
                'seq': str(record.seq),
                'quality_scores': record.letter_annotations['phred_quality']}
               for record in SeqIO.parse(f, 'fastq')]

# Create a pandas DataFrame from the list of dictionaries
df = pd.DataFrame(records)

# Print the DataFrame
print(df.head())
```

```
         seq_id                                                seq  \
0  ERR4796171.1  CAGGAACAGCAAAGGAAATCCGGCAAATTTGCGCAGTCATTCTCAA...
1  ERR4796171.2  TGGTGAATGATACCCGGTGCTGGCAATCTCGTTTAAACTACATGCA...
2  ERR4796171.3  CCACAGCATCAAGACCCTGTGACCTCTCAAAGGCCCGGTGGAAAGG...
3  ERR4796171.4  CTGGAAGTCAGACACCTGCAGATGAAGACCACAGCATCAAGACCCT...
4  ERR4796171.5  CTGGAAGTCAGACACCTGCAGATGAAGACCACAGCATCAAGACCCT...

                                      quality_scores
0  [32, 32, 32, 32, 32, 36, 36, 36, 36, 36, 36, 3...
1  [32, 32, 32, 32, 32, 36, 36, 36, 14, 32, 36, 2...
2  [32, 14, 32, 32, 32, 36, 14, 36, 36, 36, 32, 3...
3  [32, 32, 32, 32, 32, 36, 36, 36, 36, 36, 36, 3...
4  [32, 32, 21, 32, 32, 36, 32, 36, 36, 36, 36, 3...
```

Q1. Justify your model choice based on how your response is measured and any observations you may have made in your EDA.

Observations:

1) In our first observation we have observed that each sequence is having unique sequence Id. Moreover, each sequence consists of C, A, G, T Letters. The characters C, A, G, T stand for Cytosine (C), Adenine (A), Guanine (G), and Thymine (T), the four DNA nucleotides. The precise order or sequence of these nucleotides, which are the DNA molecule's building blocks, defines an organism's genetic code.
2) The ASCII values of each character are first taken from the *fastq* file before the data is converted from ASCII values to integer values. We have used `SeqIO.parse()` as follows to do the same. Here we have used `letter_annotations['phred_quality']` to get quality scores converted from ASCII values to integer.

```
# Open the FASTQ file
with open('ERR4796171.fastq', 'r') as f:
    # Parse the FASTQ file and create a list of dictionaries with the data
    records = [{'seq_id': record.id,
                'seq': str(record.seq),
                'quality_scores': record.letter_annotations['phred_quality']}
                for record in SeqIO.parse(f, 'fastq')]
```

3) The most important observation is that RNA sequence is unique for each sequence ID, causes data points to be unique. As of now, we have calculated average for all the quality scores points so that it would be easy to predict quality scores based on training and testing data.

A response variable:

A response variable that is being investigated or measured in statistics and data analysis that is anticipated to be influenced by other variables, also referred to as predictor variables or independent variables the range of ASCII values that could be used to represent quality scores in the Phred+33 encoding scheme is 33 to 126, which corresponds to Phred scores of 0 to 93. Quality scores are crucial in determining the accuracy of the sequencing data and are often used in downstream analyses such as variant calling, genome assembly, and alignment.

Response variable given before processing:

R T Q O W J F V D S Y K X H P M N B U

Response variable given after processing:

82 84 81 79 87 74 70 86 68 83 89 75 88 72 80 77 78 66 85

```
df.head()
```

	seq_id	seq	quality_scores	C	G	T	A	avg
0	ERR4796171.1	CAGGAACAGCAAAGGAAATCCGGCAAATTTGCGCAGTCATTCTCAA...	[32, 32, 32, 32, 32, 36, 36, 36, 36, 36, 36, 3...	20	17	12	26	35.73
1	ERR4796171.2	TGGTGAATGATACCCGGTGCTGGCAATCTCGTTTAAACTACATGCA...	[32, 32, 32, 32, 32, 36, 36, 36, 14, 32, 36, 2...	16	19	17	24	33.91
2	ERR4796171.3	CCACAGCATCAAGACCCTGTGACCTCTCAAAGGCCCGGTGGAAAGG...	[32, 14, 32, 32, 32, 36, 14, 36, 36, 36, 32, 3...	21	22	9	22	31.20
3	ERR4796171.4	CTGGAAGTCAGACACCTGCAGATGAAGACCACAGCATCAAGACCCT...	[32, 32, 32, 32, 32, 36, 36, 36, 36, 36, 36, 3...	22	20	10	24	35.68
4	ERR4796171.5	CTGGAAGTCAGACACCTGCAGATGAAGACCACAGCATCAAGACCCT...	[32, 32, 21, 32, 32, 36, 32, 36, 36, 36, 36, 3...	21	20	10	23	33.78

Data Processing:

Here are some data processing steps we have performed on dataset.
First, counted Adenine (A), Cytosine (C), Guanine (G), and Thymine (T) for each sequence.
And, appended additional respective columns for the same into dataframe.

```
a_c=lambda x: str(x).count('C')
df['C'] = df['seq'].apply(a_c)

a_g=lambda x: str(x).count('G')
df['G'] = df['seq'].apply(a_g)

a_t=lambda x: str(x).count('T')
df['T'] =df['seq'].apply(a_t)

a_a=lambda x: str(x).count('A')
df['A'] = df['seq'].apply(a_a)
```

Also, calculated average of the quality score for each of the sequence.

```
#df['qs'] = [ast.literal_eval(st) for st in df['quality_scores']]
df['avg'] = [round(statistics.mean(col), 2) for col in df['quality_scores']]
```

Model Choice:

While deciding model for project, first we have trained Linear Regression model as follows,

```
x = df[['C', 'G', 'T', 'A']]
y = df['avg']

x.values.reshape(4, -1)
y.values.reshape(1, -1)

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 32)

lm = LinearRegression()
lm.fit(X_train, y_train)
y_pred_train = lm.predict(X_train)
y_pred_test = lm.predict(X_test)
mse_train = mean_squared_error(y_pred_train, y_train)
mse_test = mean_squared_error(y_pred_test, y_test)

print('Intercept term: ', lm.intercept_)
print('Coefficients: \n', lm.coef_)
print('R-square, Training, Linear Regression: ', r2_score(y_train, y_pred_train))
print('R-square, Testing, Linear Regression: ', r2_score(y_test, y_pred_test))
print("MSE Train: ", mse_train)
print("MSE Test: ", mse_test)
```

Which it will give the output values as,

```
print('Intercept term: ', lm.intercept_)
print('Coefficients: \n', lm.coef_)
print('R-square, Training, Linear Regression: ', r2_score(y_train, y_pred_train))
print('R-square, Testing, Linear Regression: ', r2_score(y_test, y_pred_test))
print("MSE Train: ", mse_train)
print("MSE Test: ", mse_test)
```

```
Intercept term:  27.819542654953004
Coefficients:
 [0.08448809 0.09481963 0.10448609 0.09439769]
R-square, Training, Linear Regression:  0.014693137945421708
R-square, Testing, Linear Regression:  0.014235879454458367
MSE Train:  2.2343667424751135
MSE Test:  2.2397376161807054
```

Also, we have trained SGD (Stochastic Gradient Descent) model as also gives following output,
R-square after training SGD: 0.817
Intercept term for SGD: 2.9554296503427054e+27
Coefficients for SGD: [4.37121215e+20]
MSE for SGD: 3.3928100497380855e+49

Comparing both models, **it can be claimed that the Linear Regression model fits the provided data better** if it has a smaller mean squared error (MSE) and a higher R-squared value than the Stochastic Gradient Descent (SGD) model under comparison. A good model should have a lower MSE because it means that the pre-dictions are more likely to match the actual data. A higher R-squared value means that the model fits the data better since it explains a greater percentage of the variation in the response variable.

Q2. Report the model's test error rate using one of the techniques we discussed in lecture. Justify your choice.

Mean Squared Error:

We have calculated Mean Squared Error (MSE) by Residual Analysis. As follow,

```
lm = LinearRegression()
lm.fit(X_train, y_train)
y_pred_train = lm.predict(X_train)
y_pred_test = lm.predict(X_test)
mse_train = mean_squared_error(y_pred_train, y_train)
mse_test = mean_squared_error(y_pred_test, y_test)

print('Intercept term: ', lm.intercept_)
print('Coefficients: \n', lm.coef_)
print('R-square, Training, Linear Regression: ', r2_score(y_train, y_pred_train))
print('R-square, Testing, Linear Regression: ', r2_score(y_test, y_pred_test))
print("MSE Train: ", mse_train)
print("MSE Test: ", mse_test)
```

```
Intercept term:  27.819542654953004
Coefficients:
 [0.08448809 0.09481963 0.10448609 0.09439769]
R-square, Training, Linear Regression:  0.014693137945421708
R-square, Testing, Linear Regression:  0.014235879454458367
MSE Train:  2.2343667424751135
MSE Test:  2.2397376161807054
```

Residual analysis is a useful technique for calculating the mean squared error (MSE) in linear regres sion:

- Identifying model misspecification:
  Residual analysis allows you to diagnose the quality of the linear regression model and ident ify any systematic patterns in the errors. If there are any patterns or trends in the residual, it may indicate that the linear regression model is not capturing some of the underlying compl exity in the data.
- Evaluating model performance:
  Residual analysis can also be used to evaluate the performance of the linear regression mod el. The MSE can be calculated as the average of the squared residuals, which provides a meas ure of the accuracy of the model.

Q3. Based on the estimated test error rate, discuss how well the model fits the data.

If the estimated test error rate is low, it indicates that the model is likely to generalize well to new d ata, meaning that it is a good fit to the data. On the other hand, if the estimated test error rate is high it suggests that the model is overfitting to the training data, and may not perform well on new data.

Considering above Linear Regression model, we got low error rate indicates this model fits the data.

Q4. Use the model to make predictions for at least three cases of interest.

- Prediction 1
  For patient without migraine will have good quality scores in terms of nucleoid sequence.
  So predicted error rate should be almost zero.
  The model predicted the correct output as per requirement.

  MSE Train: **0.234**
  MSE Test: 0.039

- Prediction 2
  To predict RNA stability, using gene levels Cytosine (C), Adenine (A), Guanine (G), and Thym ine (T) based in this level we can determine presence of migraine in that particular patient.
  By using trained model, we have calculated accuracy and it came around **76%**.

```
from sklearn.metrics import mean_squared_error
import numpy as np
score = np.sqrt(mean_absolute_error(data["Actual Value"], data["Preds"]))
print("The Root Mean Squared Error of our Model is {}".format(round(score, 2)))
```

```
The Root Mean Squared Error of our Model is 0.76
```

- Prediction 3
  Based on the given gene sequence data we had plotted a boxplot graph. Looking at the graph the gene sequence having the value maximum which in our case is 32 would make an estima ted prediction of having presence of migraine in the required patient.
  Based on trained model, the **28** is the average quality score for this particular prediction.