Technion Institution

FINAL DATA ANALYSIS PROJCET

PRESENTED BY
Idan Hasdai

20/02/2025

Technion Institution

Introduction

Lets get to know the Penguins dataset from the seaborn library.The Penguins dataset includes measurements of three species—Adelie, Chinstrap, and Gentoo—collected from Palmer Archipelago, Antarctica. It contains numerical features like bill length, bill depth, flipper length, and body mass, along with categorical features like island and sex. This dataset is widely used for classification tasks, offering a real-world alternative to the classic Iris dataset, with clear yet biologically meaningful distinctions between species.

Technion Institution

imports and choosing data

```python
#imports
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix, accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.impute import KNNImputer
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.preprocessing import OneHotEncoder
```

I chose to work on the penguins dataset from the seaborn library.

```python
#loading the dataset...
df = sns.load_dataset('penguins')
```

Making sure I loaded it correctly.

```python
#cheking to see the data frame looks good
df.head(3)
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female

Exploratory Data Analysis

1 - Checking the data types of the columns.

```
#checking in what columns we have nulls and what are the data types of each column
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   species            344 non-null    object 
 1   island             344 non-null    object 
 2   bill_length_mm     342 non-null    float64
 3   bill_depth_mm      342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                333 non-null    object 
dtypes: float64(4), object(3)
memory usage: 18.9+ KB
```

2 - Checking which columns are categorical ones to encode them later on.

```
#Looking what columns i should encode
df.nunique()
```

```
species              3
island               3
bill_length_mm     164
bill_depth_mm       80
flipper_length_mm   55
body_mass_g         94
sex                  2
dtype: int64
```

3 - Checking in which columns the nulls are and how much null columns we have.

```
#checking in what columns i have nulls and how much
df.isnull().sum()
```

```
species             0
island              0
bill_length_mm      2
bill_depth_mm       2
flipper_length_mm   2
body_mass_g         2
sex                11
dtype: int64
```

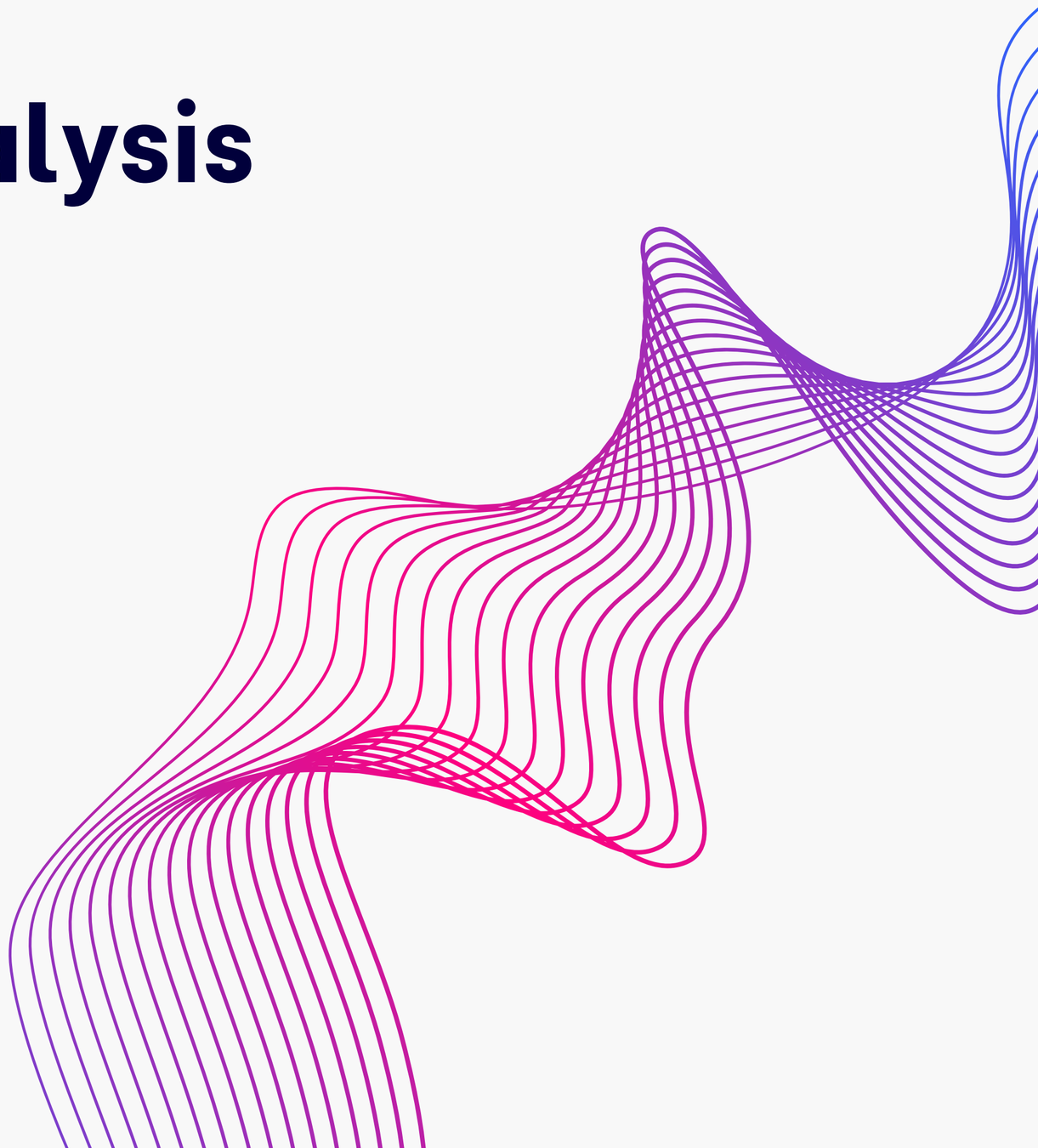
Technion Institution

Exploratory Data Analysis

Visualizing the nulls.

```
#Looking at the nulls in the data frame
df[df.isnull().any(axis=1)]
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
8	Adelie	Torgersen	34.1	18.1	193.0	3475.0	NaN
9	Adelie	Torgersen	42.0	20.2	190.0	4250.0	NaN
10	Adelie	Torgersen	37.8	17.1	186.0	3300.0	NaN
11	Adelie	Torgersen	37.8	17.3	180.0	3700.0	NaN
47	Adelie	Dream	37.5	18.9	179.0	2975.0	NaN
246	Gentoo	Biscoe	44.5	14.3	216.0	4100.0	NaN
286	Gentoo	Biscoe	46.2	14.4	214.0	4650.0	NaN
324	Gentoo	Biscoe	47.3	13.8	216.0	4725.0	NaN
336	Gentoo	Biscoe	44.5	15.7	217.0	4875.0	NaN
339	Gentoo	Biscoe	NaN	NaN	NaN	NaN	NaN



Technion Institution

Filling the NaN rows

```
#filling the NaNs in the sex column using mode(common)
df['sex'].fillna(df['sex'].mode()[0],inplace = True)
```

1 - I chose to fill the sex column with the pandas mode method,
I think it’s the best way to fill this column given that it is not a number and we can not guess what sex the penguins are.

2 - On the other hand, for the numeric columns I went with the KNN Imputer because it accounts for relationships in the data by imputing missing values based on the nearest neighbors. This approach retains patterns in the dataset, making it more accurate than simpler methods like mean or median imputation.

```
#filling the numeric NaN rows using the KNN Imputer
num_cols = ['bill_length_mm','bill_depth_mm','flipper_length_mm','body_mass_g']
imputer = KNNImputer(n_neighbors = 5)
df[num_cols] = imputer.fit_transform(df[num_cols])
```

Technion Institution

Filling the NaN rows

Checking if we have any more NaN’s we may have missed.

```
#making sure we dont have anymore nulls
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   species            344 non-null    object 
 1   island             344 non-null    object 
 2   bill_length_mm     344 non-null    float64
 3   bill_depth_mm      344 non-null    float64
 4   flipper_length_mm  344 non-null    float64
 5   body_mass_g        344 non-null    float64
 6   sex                344 non-null    object 
dtypes: float64(4), object(3)
memory usage: 18.9+ KB
```

```
#making sure again...
df.isnull().sum()
```

```
species              0
island               0
bill_length_mm       0
bill_depth_mm        0
flipper_length_mm    0
body_mass_g          0
sex                  0
dtype: int64
```

Making sure one last time.

Technion Institution

One Hot Encoding

1 - In order to be able to work on the non - numerical columns, we need to either drop them or encode them so they will be numbers that the machine can work with, because all of the non - numeric columns we have are categorical we dont need to drop them and we can One Hot Encode them, so we get a more accurate result.

```
#defining the one hot encoder tool
ohe = OneHotEncoder(sparse_output = False, drop = 'first')

#fitting the columns i want to encode to the tool
encoded = ohe.fit_transform(df[['sex','island']])

#making it back to a data fram
df_e = pd.DataFrame(encoded,columns = ohe.get_feature_names_out(['sex','island']))

#concating both the df's i made
df_code = pd.concat([df,df_e],axis = 1)

#dropping the old columns (not encoded ones)
df_code = df_code.drop(['sex','island'],axis = 1)
```

2 - I made the sparse output False so it will return an array.

3 - I used get feature names out to retrieve the names of the new features after encoding. This helps identify which columns correspond to the original categories, making it easier to work with the encoded data.

4 - After that I concated both the new and old data frames and dropped the old not encoded columns, as we don’t need them anymore.

5 - Making sure its encoded correctly.

```
#making sure its encoded
df_code.head(3)
```

	species	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex_Male	island_Dream	island_Torgersen
0	Adelie	39.1	18.7	181.0	3750.0	1.0	0.0	1.0
1	Adelie	39.5	17.4	186.0	3800.0	0.0	0.0	1.0
2	Adelie	40.3	18.0	195.0	3250.0	0.0	0.0	1.0

Label Encoding

1 - I made a Label Encoder tool to encode the target column.
The Label Encoder converts each category into a unique integer, while the OneHotEncoder creates a new binary column for each category.

```
#defining the label encoder model
le = LabelEncoder()

#fit transforming the model to the target column
df_code['species'] = le.fit_transform(df_code['species'])
```

2 - Looking both at the tail and the head of the new encoded data frame to make sure we have new numeric values for the target column (species).

```
#checking the tail of the new df to see if i label encoded it corectlly
df_code.tail(3)
```

	species	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex_Male	island_Dream	island_Torgersen
341	2	50.4	15.7	222.0	5750.0	1.0	0.0	0.0
342	2	45.2	14.8	212.0	5200.0	0.0	0.0	0.0
343	2	49.9	16.1	213.0	5400.0	1.0	0.0	0.0

```
#checking the head of the new df to make sure its labeled correctyl
df_code.head(3)
```

	species	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex_Male	island_Dream	island_Torgersen
0	0	39.1	18.7	181.0	3750.0	1.0	0.0	1.0
1	0	39.5	17.4	186.0	3800.0	0.0	0.0	1.0
2	0	40.3	18.0	195.0	3250.0	0.0	0.0	1.0

Technion Institution

Choosing Features And Target

1 - For the features variable I chose all of the columns in the data frame and dropped the target.

2 - As for the target I went with the species columns becuase I think it would be the most interesting to see if it can predict the species based on the given information.

3 - Standard Scaling - Since the dataset includes various numerical columns, each representing different measurements with distinct units, it's important to ensure they are properly weighted. This prevents differences in scale from influencing the model’s performance, allowing each feature to contribute appropriately to the classification process.

4 - The final step was to use train test and split to divide the dataset into training and testing sets. This helps evaluate the model's performance on unseen data, preventing overfitting and ensuring the model preforms well on new data.

```
#choosing the feature and target columns
X = df_code.drop(columns = ['species'],axis = 1)
y = df_code['species']

#standard scaling the feature columns so i dont have outliers
scaler = StandardScaler()
X = scaler.fit_transform(X)

#train test splitting the features and target
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2,random_state = 42)
```

Technion Institution

Choosing models and training them

```
#defining the random forest model
rf = RandomForestClassifier(n_estimators = 150,
                            random_state = 42,
                            max_depth = 15,
                            min_samples_leaf = 15,
                            min_samples_split = 15,
                            max_features = 'log2',
                            class_weight = 'balanced'
)

#defining the logistic regression model
lr = LogisticRegression()
```

1 - For this project I chose Logistic Regression and Random Forest.

2 - Training both of the models.

```
#training the logistic regression model
lr.fit(X_train,y_train)
```

```
LogisticRegression
LogisticRegression()
```

```
#training the random forest model
rf.fit(X_train,y_train)
```

```
RandomForestClassifier
RandomForestClassifier(class_weight='balanced', max_depth=15,
                       max_features='log2', min_samples_leaf=15,
                       min_samples_split=15, n_estimators=150, random_state=42)
```

After carefully changing the hyper parameters back and forth I found that for the Random Forest model these were the best hyper parameters.

As for the Logistic Regression model, after changing the hyper parameters I found that the model works the best without hyper parameters, and this gave me the best results for this model.

Technion Institution

Final Test Scores

```
#accuracy scores
rf_accuracy = accuracy_score(y_test, rf_y_pred)
lr_accuracy = accuracy_score(y_test,lr_y_pred)

print(f'Random Forest Accuracy: {rf_accuracy}')
print(f'Logistic Regression Accuracy: {lr_accuracy}')
```

```
Random Forest Accuracy: 0.9855072463768116
Logistic Regression Accuracy: 1.0
```

```
#precision scores
rf_precision = precision_score(y_test, rf_y_pred, average = 'weighted')
lr_precision = precision_score(y_test, lr_y_pred,average = 'weighted')

print(f'Random Forest Precision: {rf_precision}')
print(f'Logistic Regression Precision: {lr_precision}')
```

```
Random Forest Precision: 0.9859464207290294
Logistic Regression Precision: 1.0
```

```
#recall scores
rf_recall = recall_score(y_test, rf_y_pred,average = 'weighted')
lr_recall = recall_score(y_test, lr_y_pred,average = 'weighted')

print(f'Random Forest Recall Score: {rf_recall}')
print(f'Logistic Regression Recall Score: {lr_recall}')
```

```
Random Forest Recall Score: 0.9855072463768116
Logistic Regression Recall Score: 1.0
```

```
#f1 scores
rf_f1 = f1_score(y_test, rf_y_pred,average = 'weighted')
lr_f1 = f1_score(y_test, lr_y_pred,average = 'weighted')

print(f'Random Forest F1 Score: {rf_f1}')
print(f'Logistic Regression F1 Score: {lr_f1}')
```

```
Random Forest F1 Score: 0.9853849750062934
Logistic Regression F1 Score: 1.0
```

Predicting test scores for both models

```
#predicting the test variables for both models
rf_y_pred = rf.predict(X_test)
lr_y_pred = lr.predict(X_test)
```

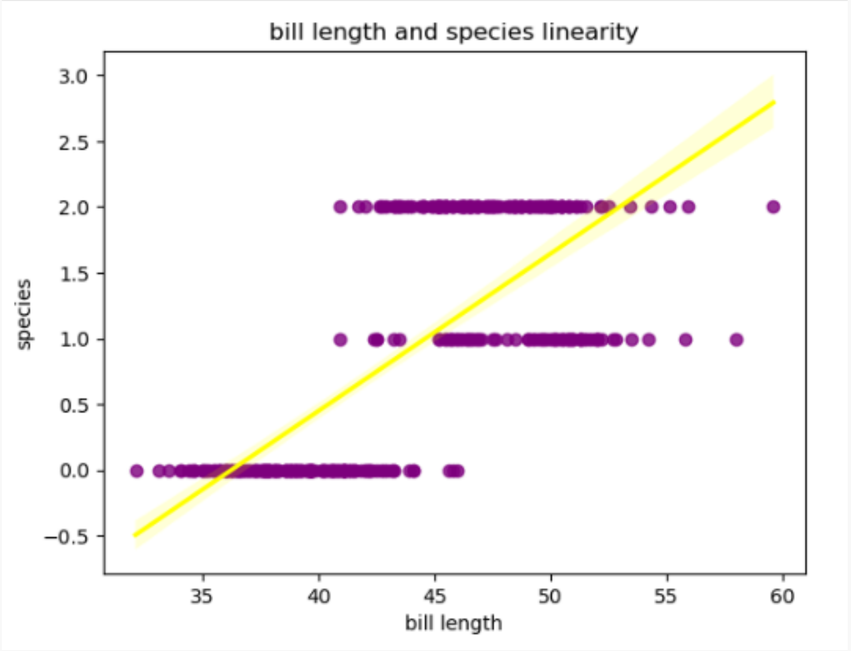
Logistic regression worked well on the penguins dataset because the data is simple and nearly linearly separable. However, this simplicity also reveals its limitations. In more complex scenarios, where relationships between features and target aren't straight lines, logistic regression might struggle. Models like random forests can capture those complex, non-linear interactions better, offering more robust and reliable performance when the data gets trickier.

I will show in the next slide how linear the relationships between a feature and the target

Technion Institution

Proving Linear Relationship

```
#making a regplot that shows the linear simplicity
sns.regplot(x = 'bill_length_mm',
            y = 'species',
            data = df_code,
            scatter_kws={'color': 'purple'},
            line_kws={'color': 'yellow'})

plt.xlabel('bill length')
plt.ylabel('species')
plt.title('bill length and species linearity')
plt.show()
```



This regplot shows how linear the data is and the reason the Logistic Regression model worked so well.
With that said, using a bigger data set which is multiclass and has more than 3 options may cause this model to struggle computing.

Technion Institution

Making sure we don’t have overfitting

```
#checking the delta between the train and the test of the Logistic Regression model to see if we have overfitting
lr_y_train = lr.predict(X_train)
Test_Accuracy = accuracy_score(y_test, lr_y_pred)
Train_Accuracy = accuracy_score(y_train, lr_y_train)
Delta = Test_Accuracy - Train_Accuracy
print('Model Performance:')
print(f" Test Accuracy: {Test_Accuracy}")
print(f" Train Accuracy: {Train_Accuracy}")
print(f' The Delta between the test and the train is: {Delta}')
```

```
Model Performance:
 Test Accuracy: 1.0
 Train Accuracy: 0.9963636363636363
 The Delta between the test and the train is: 0.0036363636363636598
```

```
#checking the delta between the train and the test of the Random Forest model to see if we have overfitting
rf_y_train = rf.predict(X_train)
Test_Accuracy = accuracy_score(y_test, rf_y_pred)
Train_Accuracy = accuracy_score(y_train, rf_y_train)
Delta = Test_Accuracy - Train_Accuracy
print('Model Performance:')
print(f" Test Accuracy: {Test_Accuracy}")
print(f" Train Accuracy: {Train_Accuracy}")
print(f' The Delta between the test and the train is: {Delta}')
```

```
Model Performance:
 Test Accuracy: 0.9855072463768116
 Train Accuracy: 1.0
 The Delta between the test and the train is: -0.01449275362318836
```

Making sure the models didn’t overfit the data and gave me a wrong answer.
I did so by checking both the test accuracy score and the train accuracy score for both models and printed the delta between them.

As we can see, the model did not over fit and actually had quite accurate results

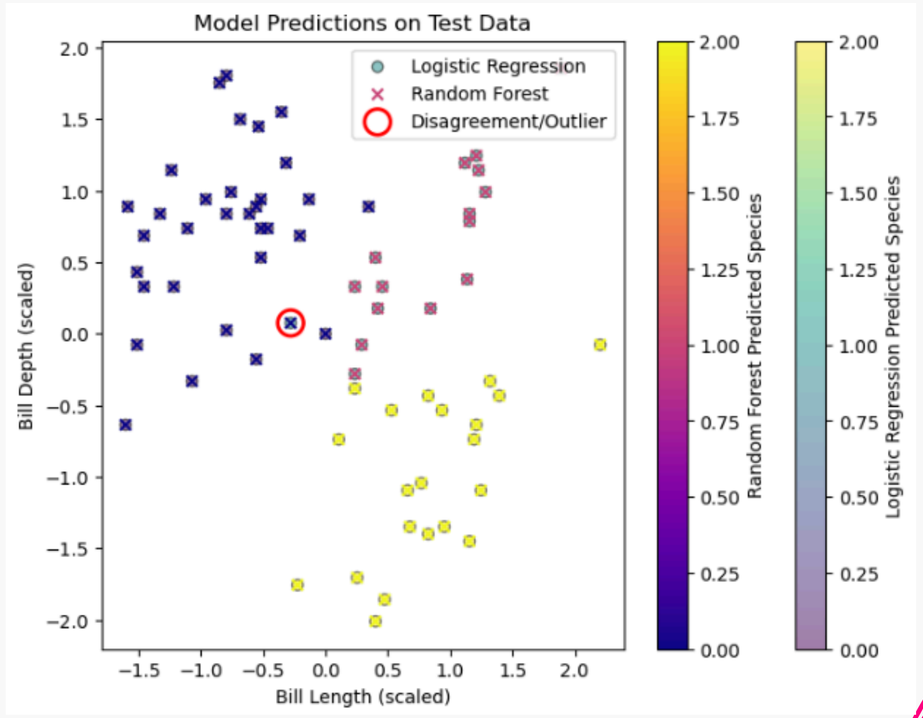
Technion Institution

Visualizing the outlier

```
plt.figure(figsize=(8, 6),dpi = 100)
mask_diff = (lr_y_pred != rf_y_pred)

#Logistic Regression predictions
sp1 = plt.scatter(X_test[:, 0],
                  X_test[:, 1],
                  c=lr_y_pred,
                  cmap='viridis',
                  edgecolor='k',
                  label='Logistic Regression',
                  alpha = 0.5)

#Random Forest predictions with a different marker and colormap
sp2 = plt.scatter(X_test[:, 0],
                  X_test[:, 1],
                  c=rf_y_pred,
                  cmap='plasma',
                  marker='x',
                  label='Random Forest')

#Making the circle to show the Disagreement
plt.scatter(X_test[mask_diff, 0],
            X_test[mask_diff, 1],
            s=200,
            facecolors='none',
            edgecolors='red',
            linewidths=2,
            label='Disagreement/Outlier')

plt.xlabel('Bill Length (scaled)')
plt.ylabel('Bill Depth (scaled)')
plt.title('Model Predictions on Test Data')

#separate colorbars for clarity
plt.colorbar(sp1, label='Logistic Regression Predicted Species')
plt.colorbar(sp2, label='Random Forest Predicted Species')
plt.legend()
plt.show()

plt.savefig('outlier scatter.jpeg')
```

1 - I made a scatter plot that shows both the Logistic Regression model and the Random Forest model prediction on the test data. In this scatter I made a circle that shows where the outlire which resulted in 98.5% accuracy score for the Random Forest model is.

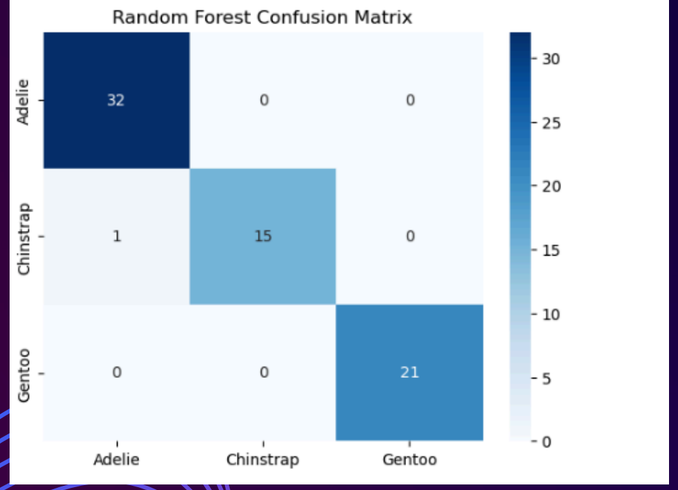
2 - This plot compares the predictions of Logistic Regression (dots) and Random Forest (X markers) on the test set. The red circle highlights a point where the models disagree. Despite this, both models perform well, indicating that the dataset is well separated. The disagreement suggests that Random Forest might be capturing more complex patterns, while Logistic Regression follows a simpler decision boundary.



Technion Institution

Confusion Matrix

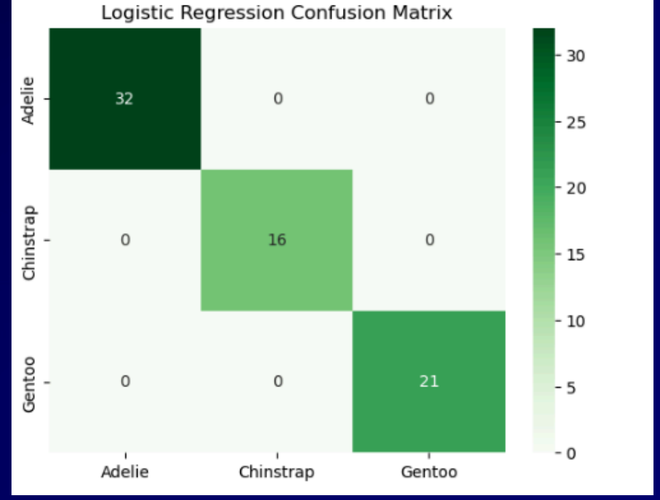
```
#defining the confusion matrix and heatmap for the Random Forest confusion matrix
rf_confusion_matrix = confusion_matrix(y_test,rf_y_pred)
sns.heatmap(rf_confusion_matrix,
            annot = True,
            fmt = 'd',
            cmap = 'Blues',
            xticklabels=le.classes_,
            yticklabels=le.classes_
)
plt.title('Random Forest Confusion Matrix')
plt.show()
```



This is the confusion matrix heatmap for the Random Forest model which shows where the model had mistaken. We can see that it predicted a chinstrap penguin as an adelie and thats the reason we got 98.5% accuracy.

This is the confusion matrix heatmap for the Logistic Regression model. We can see that differently from the Random Forest model the Logistic Regression predicted everything correctly and had zero mistakes.

```
#defining the confusion matrix and heatmap for the Logistic Regression model
lg_confusion_matrix = confusion_matrix(y_test,lr_y_pred)
sns.heatmap(lg_confusion_matrix,
            annot = True,
            fmt = 'd',
            cmap = 'Greens',
            xticklabels=le.classes_,
            yticklabels=le.classes_
)
plt.title('Logistic Regression Confusion Matrix')
plt.show()
```



Technion Institution

PROJECT CONCLUSION

This project explored the Penguins dataset, focusing on the distinctions between Adelie, Chinstrap, and Gentoo species. The data revealed clear separations between the species, making classification straightforward and highlighting their unique characteristics. By analyzing these patterns, we gain a better understanding of how species are categorized and the biological differences that set them apart. This dataset serves as a great example of how structured data can reveal meaningful insights about the natural world.

THANK YOU FOR YOUR TIME

Idan Hasdai

E-MAIL

idanhas3@gmail.com