report

April 17, 2023

0.1 Prerequisities

0.1.1 Install packages

We use imblearn library to solve imbalanced dataset by oversampling using SMOTE method.

This library can be installed by running the following cells.

[1]:
```
!pip install imblearn
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: imblearn in /usr/local/lib/python3.9/dist-
packages (0.0)
Requirement already satisfied: imbalanced-learn in
/usr/local/lib/python3.9/dist-packages (from imblearn) (0.10.1)
Requirement already satisfied: scikit-learn>=1.0.2 in
/usr/local/lib/python3.9/dist-packages (from imbalanced-learn->imblearn) (1.2.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.9/dist-
packages (from imbalanced-learn->imblearn) (1.10.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.9/dist-
packages (from imbalanced-learn->imblearn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.9/dist-packages (from imbalanced-learn->imblearn) (3.1.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.9/dist-
packages (from imbalanced-learn->imblearn) (1.22.4)
```

We use scikeras library which contains scikit-learn wrapper for Keras models.

This library can be installed by running the following cells.

[2]:
```
!pip install scikeras
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: scikeras in /usr/local/lib/python3.9/dist-
packages (0.10.0)
Requirement already satisfied: packaging>=0.21 in /usr/local/lib/python3.9/dist-
packages (from scikeras) (23.0)
Requirement already satisfied: scikit-learn>=1.0.0 in
/usr/local/lib/python3.9/dist-packages (from scikeras) (1.2.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.9/dist-
```

```
packages (from scikit-learn>=1.0.0->scikeras) (1.10.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.9/dist-packages (from scikit-learn>=1.0.0->scikeras)
(3.1.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.9/dist-
packages (from scikit-learn>=1.0.0->scikeras) (1.22.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.9/dist-
packages (from scikit-learn>=1.0.0->scikeras) (1.2.0)
```

0.1.2 Import packages

[3]:
```
from imblearn.over_sampling import SMOTENC
from numpy.random import seed
from scikeras.wrappers import KerasClassifier
from sklearn.compose import ColumnTransformer
from sklearn.metrics import classification_report, ConfusionMatrixDisplay
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
from tensorflow.keras.layers import Dense, Input, Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import set_random_seed
from tensorflow.keras.utils import plot_model
from yellowbrick.classifier import ClassPredictionError
import matplotlib.pyplot as plt
import missingno as msno
import numpy as np
import pandas as pd
import seaborn as sns
import sys
```

0.1.3 Fix randomness

[4]:
```
random_state = 42 # We use this constant in different parts of notebook.
set_random_seed(random_state) # This seeds random number in random, numpy and␣
↪tensorflow. See https://www.tensorflow.org/api_docs/python/tf/keras/utils/
↪set_random_seed
```

0.1.4 Constant definitions

[5]:
```
columns_num = ["bill_length_mm", "bill_depth_mm", "flipper_length_mm",␣
↪"body_mass_g"]
columns_cat = ["island", "sex"]
```

0.1.5 Function definitions

[6]:
```
def make_preprocessor(columns_num=columns_num, columns_cat=columns_cat):
    preprocessor_num = Pipeline(steps=[
        ("scaler", StandardScaler())
    ])
    preprocessor_cat = Pipeline(steps=[
        ("onehotencoder", OneHotEncoder())
    ])
    preprocessor = ColumnTransformer(transformers=[
        ("preprocessor_num", preprocessor_num, columns_num),
        ("preprocessor_cat", preprocessor_cat, columns_cat)
    ])
    return preprocessor


def make_pipeline(preprocessor, clf):
    return Pipeline(steps=[
        ("preprocessor", preprocessor),
        ("clf", clf)
    ])


def tune_hyperparams(clf, hyperparams, X_train, y_train,␣
↪columns_num=columns_num, columns_cat=columns_cat,␣
↪include_preprocessor=False):
    """
    Performs grid search with hyperparams on pipeline with given classifier.
    """
    preprocessor = make_preprocessor(columns_num, columns_cat)
    pipeline = make_pipeline(preprocessor, clf)
    steps = []
    if include_preprocessor:
      steps.append(("preprocessor", preprocessor))
    steps.append(("clf", clf))
    pipeline = Pipeline(steps)
    gscv = GridSearchCV(
        pipeline,
        hyperparams,
        scoring="accuracy",
        verbose=0
    )
```

```
    gscv.fit(X_train, y_train)
    print("Best params:", gscv.best_params_)
    print("Best accuracy:", gscv.best_score_)
    return gscv
```

0.2 Problem Definition

We are performing classification of penguin species.

Our target column is named "species".

Our metrics of success is "accuracy".

0.3 Data Understanding

[7]:
```
# Načtení dat
data_raw = pd.read_csv("penguins.csv")
```

[8]:
```
data_raw.head()
```

[8]:
```
  species     island  bill_length_mm  bill_depth_mm  flipper_length_mm  \
0  Adelie  Torgersen            39.1           18.7              181.0
1  Adelie  Torgersen            39.5           17.4              186.0
2  Adelie  Torgersen            40.3           18.0              195.0
3  Adelie  Torgersen             NaN            NaN                NaN
4  Adelie  Torgersen            36.7           19.3              193.0

   body_mass_g     sex  year
0       3750.0    male  2007
1       3800.0  female  2007
2       3250.0  female  2007
3          NaN     NaN  2007
4       3450.0  female  2007
```

Our dataset contains:

- 1 column with target variable (`species`)
- 7 columns with possible input features (`island`, `bill_length_mm`, `bill_depth_mm`, `flipper_length_mm`, `body_mass_g`, `sex`, `year`)
- 5 columns with numerical values (`bill_length_mm`, `bill_depth_mm`, `flipper_length_mm`, `body_mass_g`, `year`), out of these 4 contains floating numbers (`bill_length_mm`, `bill_depth_mm`, `flipper_length_mm`, `body_mass_g`) and 1 contain integer number (`year`)
- 3 columns with categorical values (`species`, `island`, `sex`)

[9]:
```
data_raw["species"].value_counts()
```

[9]:
```
Adelie       160
Gentoo       124
Chinstrap     79
Name: species, dtype: int64
```

As a result of calling `value_counts()` on our target column shows:

- We are performing a multi-class classification.
- Class counts are imbalanced, therefore we need to solve class imbalance during training. We have decided to use oversampling using SMOTE method to solve class imbalance in our case.

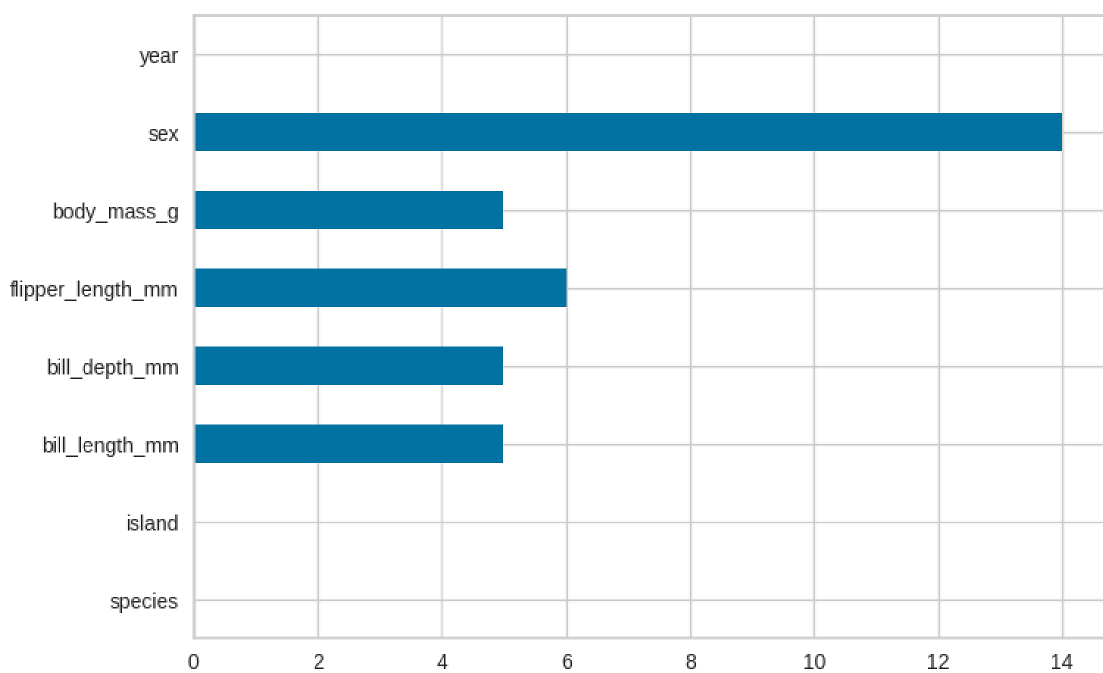
0.3.1 Indication of missing values

[10]:
```
data_raw.isnull().sum().to_frame("number of null values")
```

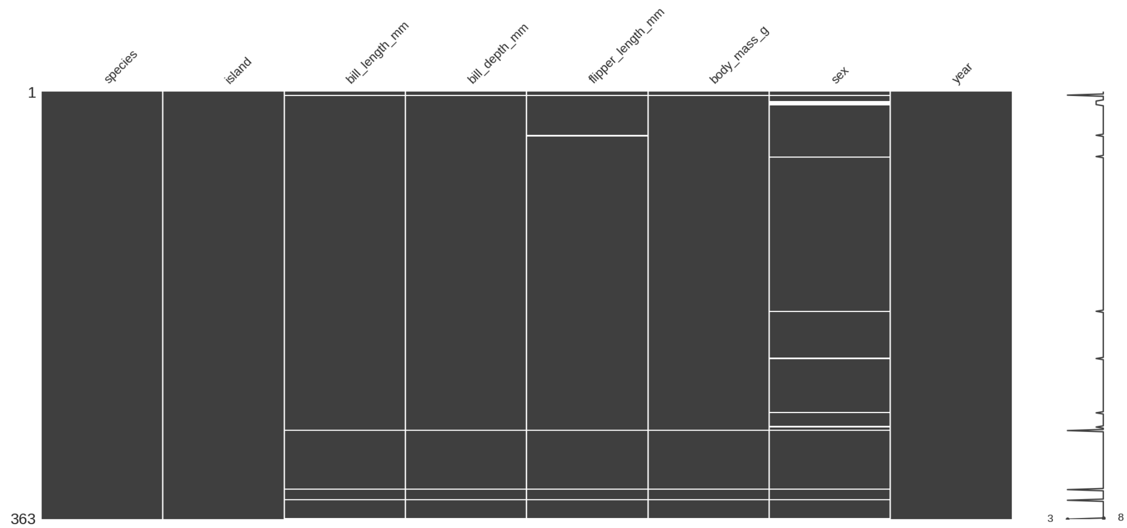
[10]:

	number of null values
species	0
island	0
bill_length_mm	5
bill_depth_mm	5
flipper_length_mm	6
body_mass_g	5
sex	14
year	0

[11]:
```
data_raw.isnull().sum().plot(kind="barh");
```



[12]:
```
msno.matrix(data_raw)
plt.savefig("fig_missing_values.png")
plt.show()
```



Our dataset contains a small amount of missing values in columns `bill_length_mm`, `bill_depth_mm`, `flipper_length_mm`, `body_mass_g`, `sex`.

0.3.2 Descriptive statistics

[13]:
```
print("Dataset contains {} rows and {} columns.".format(*data_raw.shape))
```

```
Dataset contains 363 rows and 8 columns.
```

[14]:
```
data_raw.describe(include = 'all')
```

[14]:
```
        species  island  bill_length_mm  bill_depth_mm  flipper_length_mm  \
count       363     363      358.000000     358.000000         357.000000
unique        3       3             NaN            NaN                NaN
top      Adelie  Biscoe             NaN            NaN                NaN
freq        160     170             NaN            NaN                NaN
mean        NaN     NaN       43.926257      17.205587         200.451261
std         NaN     NaN        5.441240       1.951749          14.000754
min         NaN     NaN       32.100000      13.100000         172.000000
25%         NaN     NaN       39.350000      15.700000         190.000000
50%         NaN     NaN       44.450000      17.500000         197.000000
75%         NaN     NaN       48.500000      18.700000         213.000000
max         NaN     NaN       59.600000      21.500000         231.000000

        body_mass_g     sex         year
count    358.000000     349   363.000000
unique          NaN       2          NaN
top             NaN  female          NaN
freq            NaN     175          NaN
mean    4173.743017     NaN  2007.991736
```

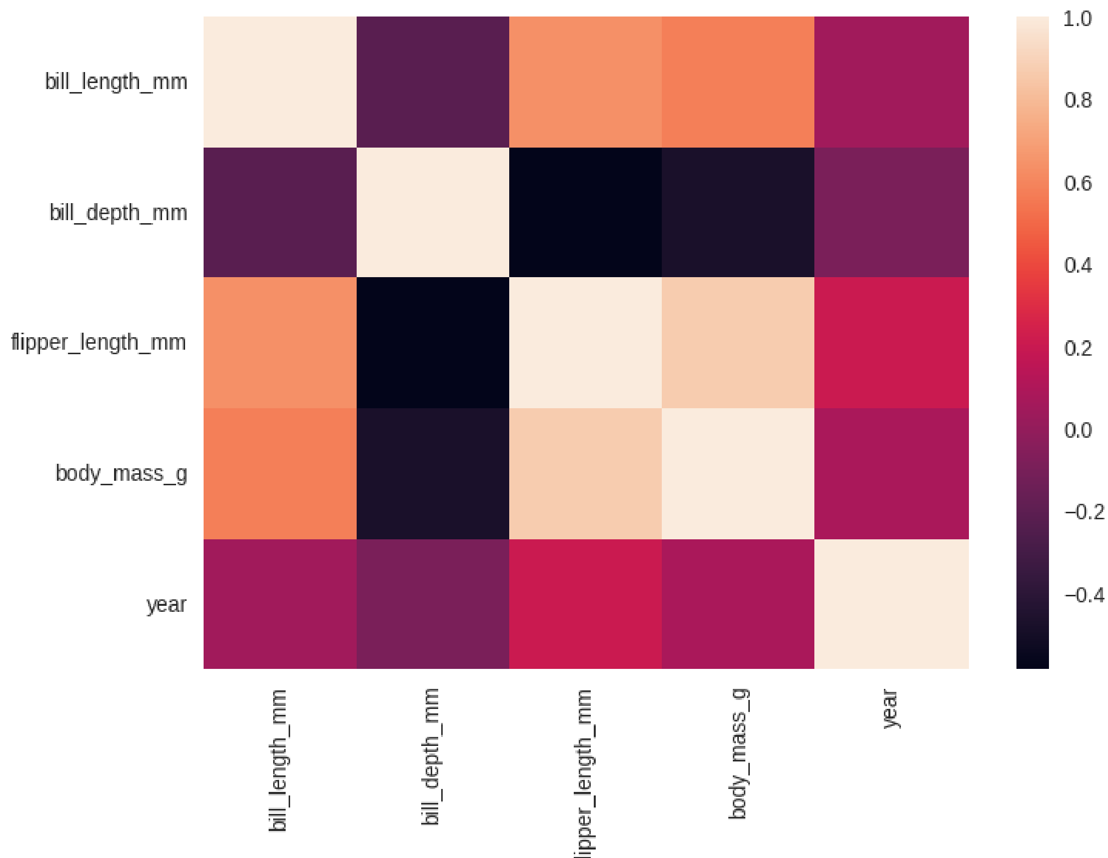
```
std     796.395388    NaN     0.829323
min    2700.000000    NaN  2007.000000
25%    3550.000000    NaN  2007.000000
50%    3950.000000    NaN  2008.000000
75%    4743.750000    NaN  2009.000000
max    6300.000000    NaN  2009.000000
```

0.3.3 Correlation analysis

[15]:

```
sns.heatmap(data_raw.corr())
plt.savefig("fig_correlation.png")
plt.show()
```

```
<ipython-input-15-2f9a9ba54672>:1: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  sns.heatmap(data_raw.corr())
```



0.3.4 Duplicates

[16]:
```
print("Number of duplicate rows to delete:", data_raw[data_raw.
 ↪duplicated(keep="first")].shape[0])
```

```
Number of duplicate rows to delete: 13
```

0.4 Data Preparation

We would like to make a decision about penguin species based on physical characteristics of the penguin and on which island he has been found. That should both be available at the time of prediction.

We would not like to make a prediction of penguin species based on year as years change and we might want to make a prediction of penguin species in future, where year can be some value not found in training data. That would degrade the quality of predictions. Therefore we remove year column.

[17]:
```
def preprocess_remove_year(df):
    """
    Removes year column.
    """
    df.drop(columns="year", inplace=True)


def preprocess_remove_multiple_NaNs_rows(df):
    """
    Removes rows containing 5 or more NaN rows.
    """
    df.drop(index=df[df.isnull().sum(axis = 1) >= 5].index, inplace=True)


def preprocess_remove_duplicates(df):
    """
    Removes duplicate rows, but keep first row of duplicate rows.
    """
    df.drop(index=df[df.duplicated(keep="first")].index, inplace=True)


def preprocess_convert_datatypes(df):
    """
    Performs datatype conversions.
    """
    df["species"] = df["species"].astype("category")
    df["island"] = df["island"].astype("category")
    df["sex"] = df["sex"].astype("category")


def preprocess_fillna_sex_by_unknown_value(df):
```

```
    """
    Fills in missing values of sex column
    by filling in a value "unknown".
    """
    df["sex"] = df["sex"].astype("str").fillna("unknown").astype("category")


def preprocess_fillna_sex_by_random_forest_model(df):
    """
    Fills in missing values of sex column
    by training a random forest model on rows, where sex column value is filled
    and using that model to predict value of sex in rows,
    where sex column value is missing.
    """
    df_sex_isna = df[df["sex"].isnull()]
    df_sex_nona = df[df["sex"].notnull()]
    sex_y = df_sex_nona["sex"]
    sex_X = df_sex_nona.drop(columns=["sex"])
    random_forest_model = RandomForestClassifier(random_state=random_state)
    random_forest_hyperparams = {
        'clf__n_estimators': [50], #, 100, 150, 200],
        'clf__criterion': ['gini', 'entropy'],
        'clf__max_depth': [None, 10, 20, 30],
        'clf__min_samples_split': [2, 5, 10],
        'clf__min_samples_leaf': [1, 2, 4]
    }
    columns_num = ["bill_length_mm", "bill_depth_mm", "flipper_length_mm",␣
 ↪"body_mass_g"]
    columns_cat = ["island"]
    gscv = tune_hyperparams(random_forest_model, random_forest_hyperparams,␣
 ↪sex_X, sex_y, columns_num, columns_cat, include_preprocessor=True)
    sex_predictions = gscv.predict(df_sex_isna)
    df.loc[df["sex"].isnull(), "sex"] = sex_predictions


def preprocess_fillna(df):
    """
    Fills NaN values.
    """
    df["bill_length_mm"].fillna(df["bill_length_mm"].mean(), inplace=True)
    df["bill_depth_mm"].fillna(df["bill_depth_mm"].mean(), inplace=True)
    df["flipper_length_mm"].fillna(df["flipper_length_mm"].mean(), inplace=True)
    df["body_mass_g"].fillna(df["body_mass_g"].mean(), inplace=True)

def preprocess(df, for_training=True):
    """
    Preprocesses dataframe df and returns preprocessed dataframe.
```

```
    """
    df_preprocessed = df.copy()
    preprocess_remove_year(df_preprocessed)
    preprocess_remove_multiple_NaNs_rows(df_preprocessed)
    preprocess_remove_duplicates(df_preprocessed)
    preprocess_convert_datatypes(df_preprocessed)

    y = df_preprocessed["species"]
    X = df_preprocessed.drop(columns=["species"])

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=random_state, stratify=y)

    preprocess_fillna(X_train)
    preprocess_fillna(X_test)

    preprocess_fillna_sex_by_random_forest_model(X_train)
    preprocess_fillna_sex_by_random_forest_model(X_test)

    #preprocess_fillna_sex_by_unknown_value(X_train)
    #preprocess_fillna_sex_by_unknown_value(X_test)

    # SMOTENC documentation: https://imbalanced-learn.org/stable/references/
 ↪generated/imblearn.over_sampling.SMOTENC.html
    if for_training:
        X_train, y_train = SMOTENC([0, 5], random_state=random_state).
 ↪fit_resample(X_train, y_train)

    preprocessor = make_preprocessor()
    X_train_transformed = preprocessor.fit_transform(X_train)
    X_test_transformed = preprocessor.transform(X_test)
    X_train = pd.DataFrame(X_train_transformed, index=X_train.index,␣
 ↪columns=preprocessor.get_feature_names_out())
    X_test = pd.DataFrame(X_test_transformed, index=X_test.index,␣
 ↪columns=preprocessor.get_feature_names_out())

    return X_train, X_test, y_train, y_test

X_train, X_test, y_train, y_test = preprocess(data_raw)
```

```
Best params: {'clf__criterion': 'gini', 'clf__max_depth': None,
'clf__min_samples_leaf': 2, 'clf__min_samples_split': 2, 'clf__n_estimators':
50}
Best accuracy: 0.8923130677847659
Best params: {'clf__criterion': 'gini', 'clf__max_depth': None,
'clf__min_samples_leaf': 2, 'clf__min_samples_split': 5, 'clf__n_estimators':
50}
```

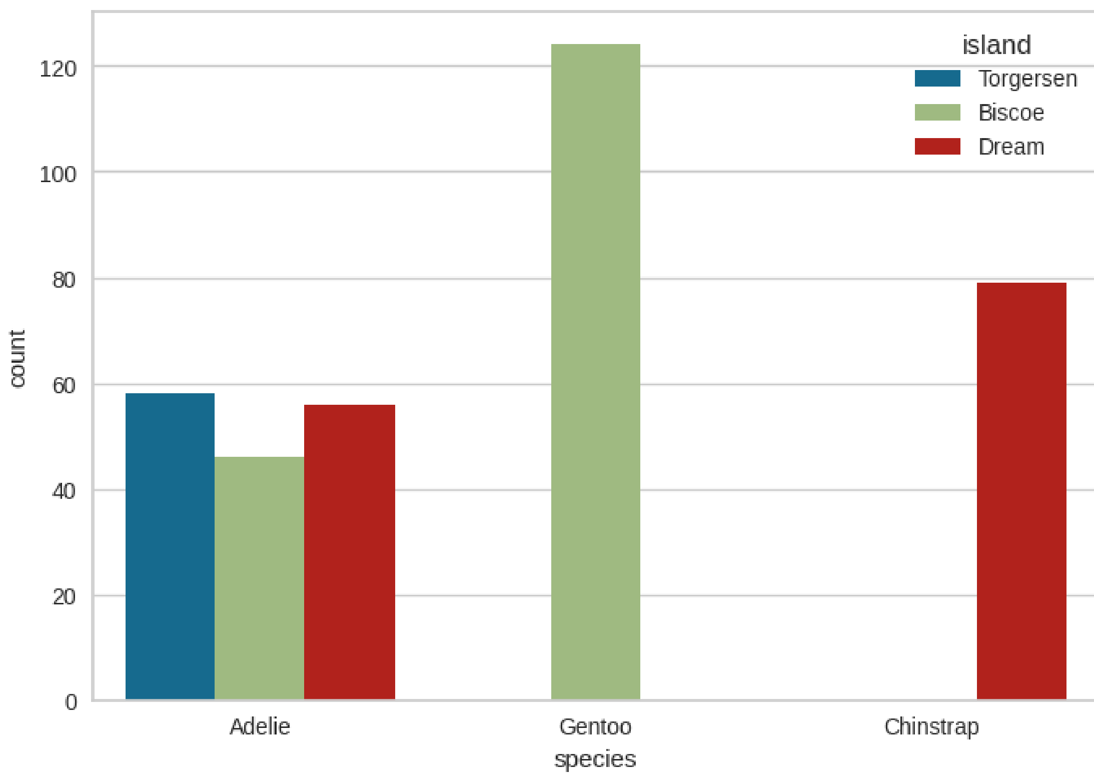
```
Best accuracy: 0.9120879120879121
```

[18]:
```
X_train.to_csv("X_train.csv")
X_test.to_csv("X_test.csv")
y_train.to_csv("y_train.csv")
y_test.to_csv("y_test.csv")
```

0.5 Data Visualization

0.5.1 Island habitability

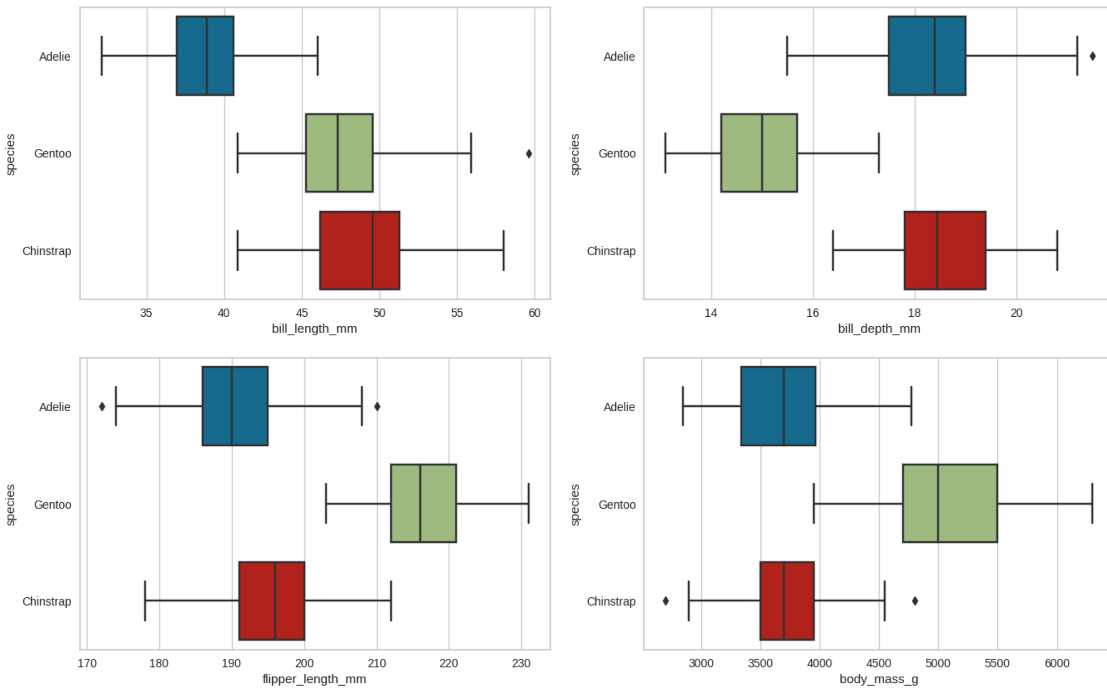
[19]:
```
sns.countplot(data_raw, x='species', hue='island')
plt.savefig("fig_island_habitability.png")
plt.show()
```



- Adelie species are found on all three islands, but Gentoo and Chinstrap are separated.
- Gentoo lives on Biscoe Island
- Chinstrap lives on Dream Island

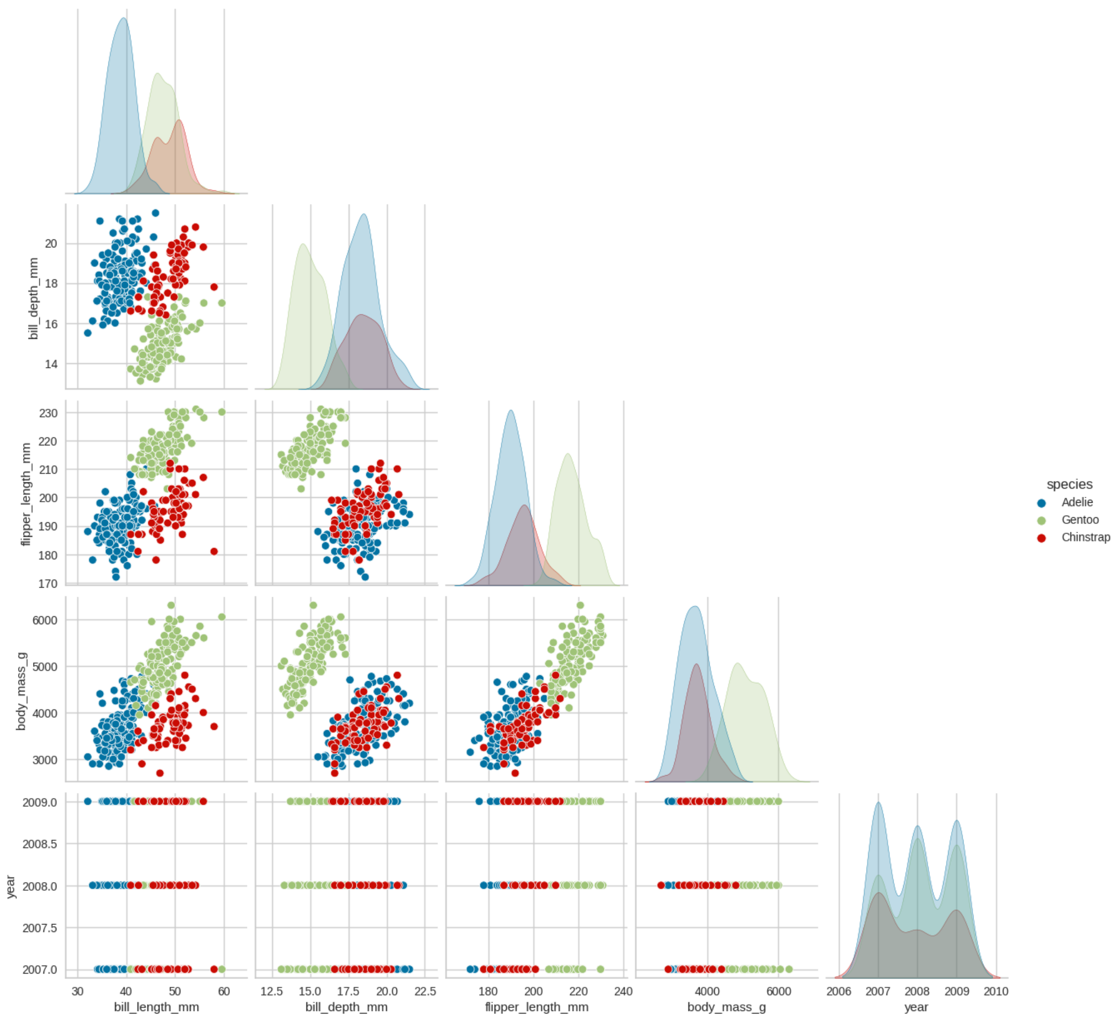
0.5.2 Boxplots

[20]:
```
fig, ax = plt.subplots(2, 2, figsize=(16, 10))
sns.boxplot(x="bill_length_mm", y="species", data=data_raw, ax=ax[0,0])
sns.boxplot(x="bill_depth_mm", y="species", data=data_raw, ax=ax[0, 1])
sns.boxplot(x="flipper_length_mm", y="species", data=data_raw, ax=ax[1,0])
sns.boxplot(x="body_mass_g", y="species", data=data_raw, ax=ax[1,1])
plt.savefig("fig_boxplots.png")
plt.show()
```



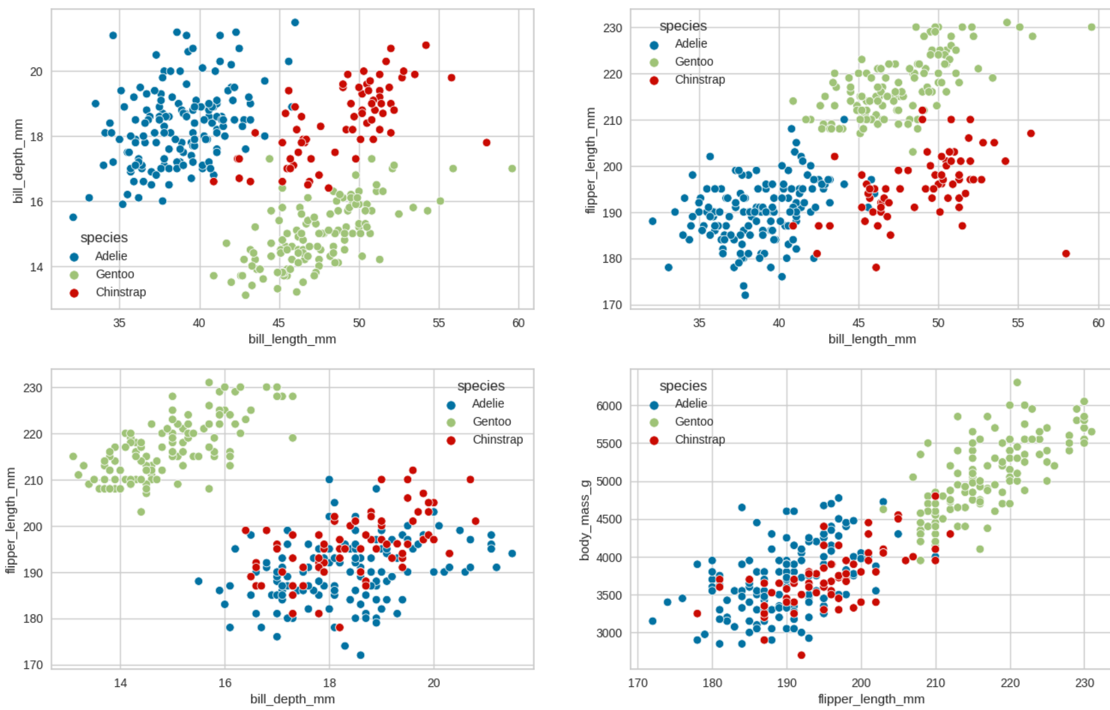
0.5.3 Pairplot

[21]:
```
sns.pairplot(data_raw, hue = 'species', corner=True)
plt.savefig("fig_pairplot.png")
plt.show()
```



0.5.4 Scatterplots

[22]:
```
fig, ax = plt.subplots(2, 2, figsize=(16, 10))
sns.scatterplot(x="bill_length_mm", y="bill_depth_mm", hue="species",␣
  ↪data=data_raw, ax=ax[0,0])
sns.scatterplot(x="bill_length_mm", y="flipper_length_mm", hue="species",␣
  ↪data=data_raw, ax=ax[0,1])
sns.scatterplot(x="bill_depth_mm", y="flipper_length_mm", hue="species",␣
  ↪data=data_raw, ax=ax[1,0])
sns.scatterplot(x="flipper_length_mm", y="body_mass_g", hue="species",␣
  ↪data=data_raw, ax=ax[1,1])
plt.savefig("fig_scatterplots.png")
plt.show()
```

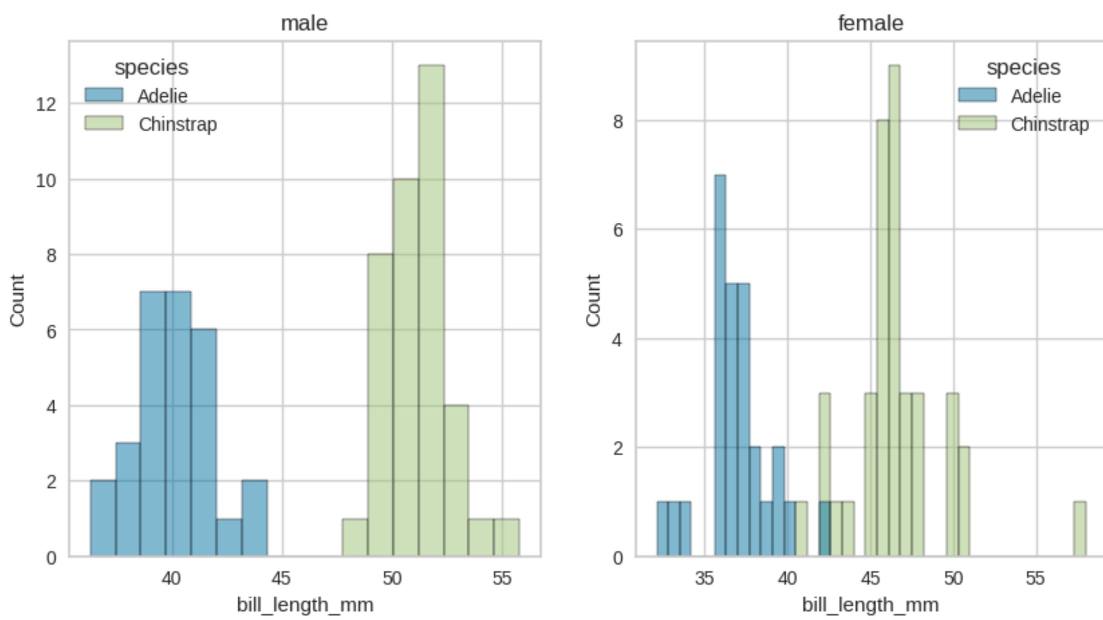


0.5.5 Adelie vs Chinstrap

Let's try to find the differences between Adelie and Chinstrap We will use the bill_lenth_mm attribute for this, since it is the only attribute that shows significant differences based on the graph of pairwise dependencies.

[23]:
```
# first, let's prepare filtered data samples by sex and species (Adelie is additionally filtered to Dream Island, because Chinstrap lives only on it)
adelie_male = data_raw[(data_raw['species'] == 'Adelie') & (data_raw['sex'] == 'male') & (data_raw['island'] == 'Dream')]
adelie_female = data_raw[(data_raw['species'] == 'Adelie') & (data_raw['sex'] == 'female') & (data_raw['island'] == 'Dream')]

chinstrap_male = data_raw[(data_raw['species'] == 'Chinstrap') & (data_raw['sex'] == 'male')]
chinstrap_female = data_raw[(data_raw['species'] == 'Chinstrap') & (data_raw['sex'] == 'female')]

# two histograms describing the distribution of the bill_lenth_mm attribute based on gender
adelie_chinstrap_male = pd.concat([adelie_male, chinstrap_male])
adelie_chinstrap_female = pd.concat([adelie_female, chinstrap_female])

plt.figure(figsize = (10,5))
```

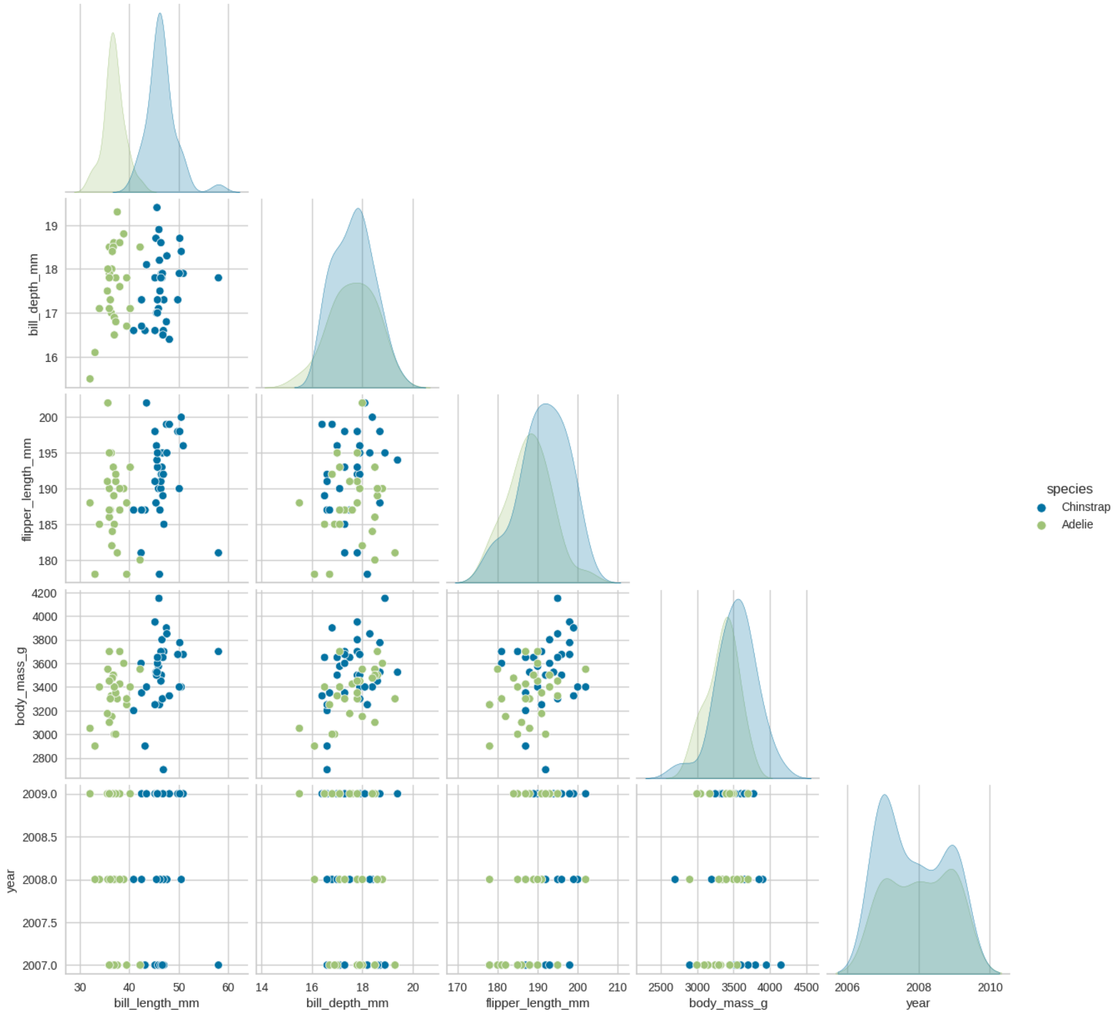
```
plt.subplot(1, 2, 1)
sns.histplot(adelie_chinstrap_male, x = 'bill_length_mm', hue = 'species',␣
↪element = 'bars', bins = 17)
plt.title('male')

plt.subplot(1, 2, 2)
sns.histplot(adelie_chinstrap_female, x = 'bill_length_mm', hue = 'species',␣
↪bins = 37)
plt.title('female')
plt.savefig("fig_adelie_chinstrap.png")
plt.show()
```



We can see that Adelie and Chinstrap can be easily separeted if sex = male, but not so easily if value of sex = female.

Let's look at the female gender in more detail and use a graph of pairwise dependencies.

[24]:
```
df_adelie_chinstrap_female = pd.concat([chinstrap_female, adelie_female])
sns.pairplot(df_adelie_chinstrap_female, hue= 'species', corner=True)
plt.savefig("fig_adelie_chinstrap2.png")
plt.show()
```



The best linear separation on the surface we can observe with the attributes x = bill_length_mm y = flipper_length_mm

0.6 Modeling

We decided to try few different models, mainly the ones mentioned in class. Therefore we have tried:

- Support vector classifier
- Tree model
- Random Forest model
- Neural Network model (in Keras using Scikit-learn compatible wrapper)
- Bagging classifier

[25]:
```
accuracies = []
```

0.6.1 Model support vector classifier

[26]:
```
linearsvc_model = LinearSVC(random_state=random_state)
linearsvc_hyperparams = {
    "clf__C": [0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
}

gscv_linearsvc = tune_hyperparams(linearsvc_model, linearsvc_hyperparams,␣
 ↪X_train, y_train)
accuracies.append(["linearsvc", gscv_linearsvc.best_score_])
```

```
Best params: {'clf__C': 0.5}
Best accuracy: 0.9972602739726029
```

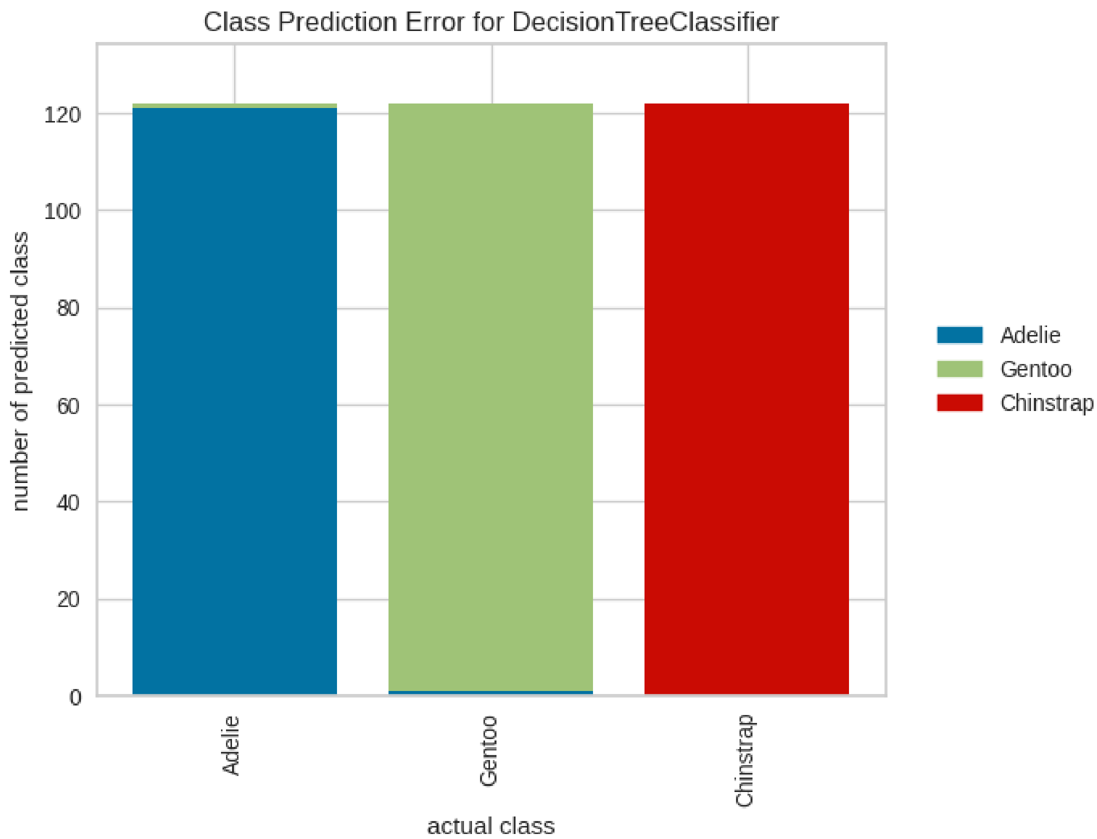
0.6.2 Model tree

[27]:
```
tree_model = DecisionTreeClassifier(random_state=random_state)
tree_hyperparams = {
    "clf__criterion": ['gini', 'entropy'],
    "clf__max_depth": [2,4,6,8,10,12]
}

gscv_tree = tune_hyperparams(tree_model, tree_hyperparams, X_train, y_train)
accuracies.append(["tree", gscv_tree.best_score_])
```

```
Best params: {'clf__criterion': 'gini', 'clf__max_depth': 4}
Best accuracy: 0.9781562384302112
```

[28]:
```
# Instantiate the classification model and visualizer
visualizer = ClassPredictionError(
    gscv_tree.best_estimator_, classes=y_train.unique()
)

visualizer.score(X_train, y_train)
visualizer.show()
```

```
/usr/local/lib/python3.9/dist-packages/yellowbrick/classifier/base.py:232:
YellowbrickWarning: could not determine class_counts_ from previously fitted
classifier
  warnings.warn(
```



[28]:
```
<Axes: title={'center': 'Class Prediction Error for DecisionTreeClassifier'},
xlabel='actual class', ylabel='number of predicted class'>
```

0.6.3 Model random forest

[29]:
```
random_forest_model = RandomForestClassifier(random_state=random_state)
random_forest_hyperparams = {
    'clf__n_estimators': [50], #, 100, 150, 200],
    'clf__criterion': ['gini', 'entropy'],
    'clf__max_depth': [None, 10, 20, 30],
    'clf__min_samples_split': [2, 5, 10],
    'clf__min_samples_leaf': [1, 2, 4]
}
gscv_rfr = tune_hyperparams(random_forest_model, random_forest_hyperparams,␣
↪X_train, y_train)
accuracies.append(["random forest", gscv_rfr.best_score_])
```

```
Best params: {'clf__criterion': 'gini', 'clf__max_depth': None,
'clf__min_samples_leaf': 1, 'clf__min_samples_split': 5, 'clf__n_estimators':
50}
Best accuracy: 0.9890781192151055
```

0.6.4 Model bagging classifier

[30]:
```
decision_tree_model = DecisionTreeClassifier(random_state=random_state,␣
 ↪criterion="gini", max_depth=4)
bagging_model = BaggingClassifier(random_state=random_state,␣
 ↪estimator=decision_tree_model)

bagging_hyperparams = {
    "clf__max_features": [0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
    "clf__max_samples": [0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
}

gscv_bagging = tune_hyperparams(bagging_model, bagging_hyperparams, X_train,␣
 ↪y_train)
accuracies.append(["bagging", gscv_bagging.best_score_])
```

```
Best params: {'clf__max_features': 0.6, 'clf__max_samples': 0.6}
Best accuracy: 0.9890781192151055
```

0.6.5 Model neural network

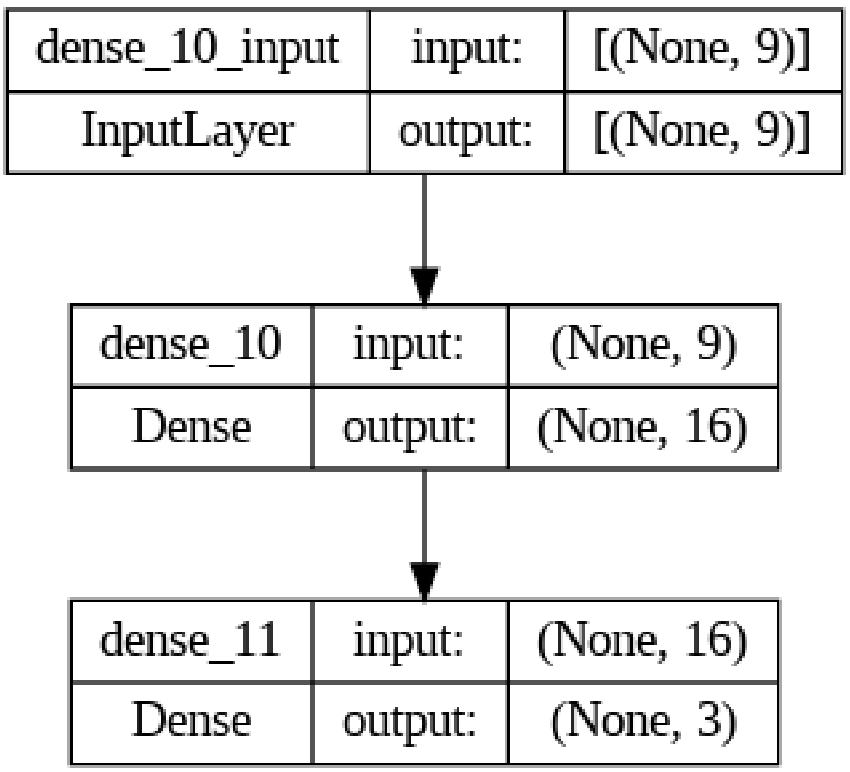
[31]:
```
num_input_features = X_train.shape[1]
num_predicted_classes = y_train.nunique()

def make_model():
    model = Sequential()
    model.add(Dense(16, input_dim=num_input_features, activation="relu"))
    model.add(Dense(num_predicted_classes, activation="softmax"))
    model.compile(
        loss="sparse_categorical_crossentropy",
        optimizer="adam",
        metrics=["accuracy"]
    )
    return model

neural_network_model = KerasClassifier(model=make_model, epochs=50, verbose=0,␣
 ↪random_state=random_state)
neural_network_hyperparams = {}
gscv_neural_network = tune_hyperparams(neural_network_model,␣
 ↪neural_network_hyperparams, X_train, y_train)
accuracies.append(["neural_network", gscv_neural_network.best_score_])
```

```
WARNING:tensorflow:5 out of the last 13 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x7f04c17549d0>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
```

```
@tf.function has reduce_retracing=True option that can avoid unnecessary
retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.

Best params: {}
Best accuracy: 1.0
```

[32]:
```python
plot_model(gscv_neural_network.best_estimator_["clf"].model_,show_shapes=True,␣
 ↪show_layer_names=True, to_file="fig_neural_network_model.png")
plot_model(gscv_neural_network.best_estimator_["clf"].model_,show_shapes=True,␣
 ↪show_layer_names=True)
```

[32]:



0.6.6 Model accuracies

[33]:
```python
df_accuracies = pd.DataFrame(data=accuracies, columns=["model", "accuracy"])
df_accuracies.sort_values(by="accuracy", ascending=False)
```

[33]:

	model	accuracy
4	neural_network	1.000000
0	linearsvc	0.997260
2	random forest	0.989078
3	bagging	0.989078

```
1            tree  0.978156
```

0.6.7 Conclusion on selecting the best model

As we can see, the best accuracy has neural network. That is great also because the assignment explicitly mentions we must save model in `h5` format, and we know an easy way how to do it in Keras (which we use in our neural network model) described here for Keras and here in Chapter 4.2 for scikeras, so we have decided to select neural network as the best model.

Our model is limited by following factors:

- The architecture is limited by number of input features, which places a restriction on the number of neurons in input layer, which must be same as number of input features. We do this by setting `input_dim` parameter of first `Dense` layer, however we probably could achieve the same result by using separate `Input` layer with required number of neurons.
- The architecture is limited by performed task, which is a multi class classification, which places the following restrictions on the network: a) the number of neurons in output layer must be the same as number of predicted classes b) we must use softmax activation function on the last layer, which returns a sequence of predicted class probability scores between `0` and `1` which summs up to `1`, where the class with maximum probability score is the class which should be predicted.

Model could be improved by following ways:

- Using `Dropout` layer would prevent overfitting.

Values of hyperparameters (such as number of layers or number of neurons in layer) are hardcoded in our `make_model()` function and were chosen in such a way that we wanted to start with some small number of layers and neurons and if it would not work well, increase the number of layers or neurons. However, because chosen values worked well, we have not needed to use more complicated networks.

0.6.8 Save and load trained model

[34]:
```
gscv_neural_network.best_estimator_["clf"].model_.save("model.h5")
```

[35]:
```
def load(path):
    keras_model = load_model(path)
    scikeras_model = KerasClassifier(keras_model)
    scikeras_model.initialize(X_train, y_train)
    pipeline = Pipeline(steps=[
        ("clf", scikeras_model)
    ])
    return pipeline

best_model = load("model.h5")
```

0.7 Evaluation

0.7.1 Make predictions on test set using best model

[36]:
```
y_pred = best_model.predict(X_test)
```

```
WARNING:tensorflow:5 out of the last 13 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x7f04b84a89d0>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid unnecessary
retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

```
3/3 [==============================] - 0s 4ms/step
```

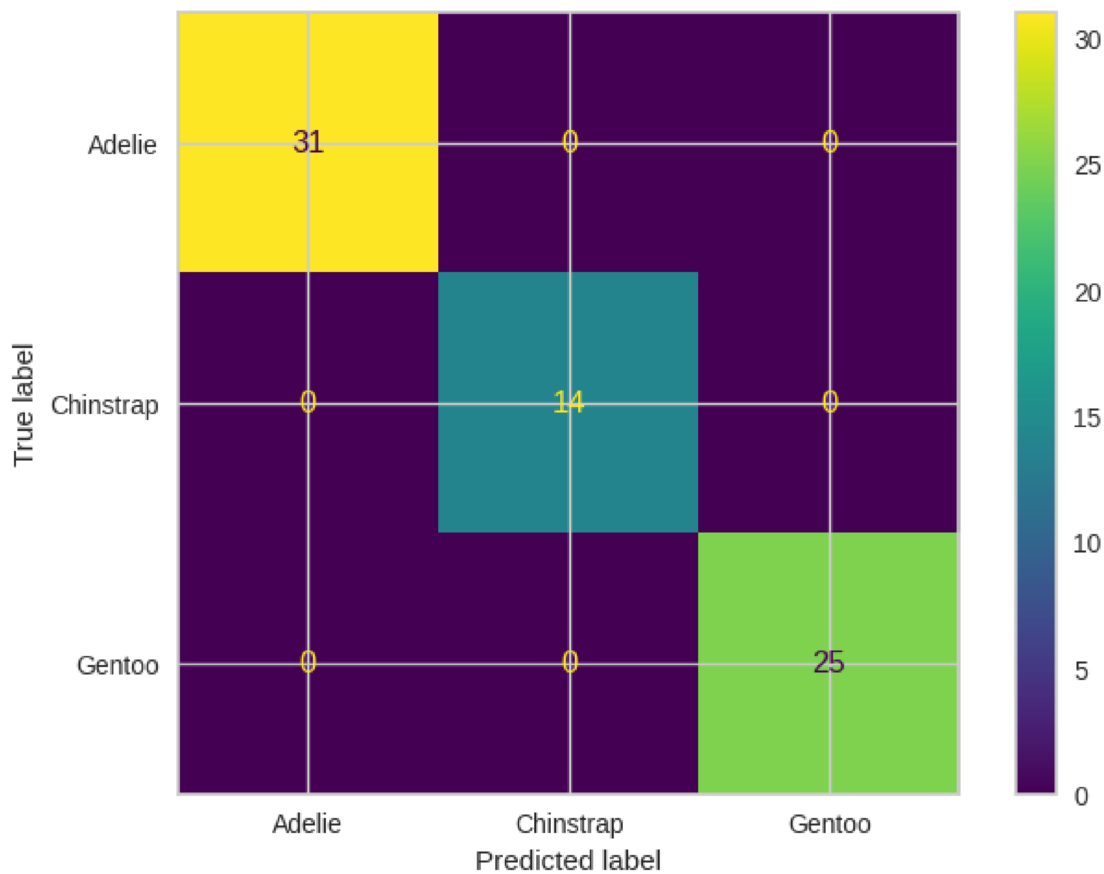
0.7.2 Show classification report

[37]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

      Adelie       1.00      1.00      1.00        31
   Chinstrap       1.00      1.00      1.00        14
      Gentoo       1.00      1.00      1.00        25

    accuracy                           1.00        70
   macro avg       1.00      1.00      1.00        70
weighted avg       1.00      1.00      1.00        70
```

0.7.3 Show confusion matrix

[38]:
```
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.savefig("fig_confusion_matrix.png")
plt.show()
```



As we can see from classification report and confusion matrix, our model achieves 100% accuracy.

0.8 Development Environment Characteristics

0.8.1 Python version

We use the following Python version:

[39]:
```
print(sys.version)
```

```
3.9.16 (main, Dec  7 2022, 01:11:51)
[GCC 9.4.0]
```

0.8.2 Packages utilized and their versions

We directly use the following packages:

- `imblearn` - for oversampling using SMOTE method
- `numpy` - for seeding random number for Keras
- `matplotlib` - for visualization
- `pandas` - for working with data in table form
- `scikeras` - contains scikit-learn compatible wrapper which we use

- `seaborn` - for visualization
- `scikit-learn` - for non-neural network models
- `tensorflow` - for neural network models using Keras API
- `yellowbrick` - for visualization of class prediction error

The full list of packages installed in our Google Collab environment and their versions can be found in the following commands output:

[40]:
```
!pip freeze
```

```
absl-py==1.4.0
alabaster==0.7.13
albumentations==1.2.1
altair==4.2.2
anyio==3.6.2
appdirs==1.4.4
argon2-cffi==21.3.0
argon2-cffi-bindings==21.2.0
arviz==0.15.1
astropy==5.2.2
astunparse==1.6.3
attrs==22.2.0
audioread==3.0.0
autograd==1.5
Babel==2.12.1
backcall==0.2.0
beautifulsoup4==4.11.2
bleach==6.0.0
blis==0.7.9
blosc2==2.0.0
bokeh==2.4.3
branca==0.6.0
CacheControl==0.12.11
cached-property==1.5.2
cachetools==5.3.0
catalogue==2.0.8
certifi==2022.12.7
cffi==1.15.1
chardet==4.0.0
charset-normalizer==2.0.12
chex==0.1.7
click==8.1.3
cloudpickle==2.2.1
cmake==3.25.2
cmdstanpy==1.1.0
colorcet==3.0.1
colorlover==0.3.0
community==1.0.0b1
confection==0.0.4
```

```
cons==0.4.5
contextlib2==0.6.0.post1
contourpy==1.0.7
convertdate==2.4.0
cryptography==40.0.1
cufflinks==0.17.3
cvxopt==1.3.0
cvxpy==1.3.1
cycler==0.11.0
cymem==2.0.7
Cython==0.29.34
dask==2022.12.1
datascience==0.17.6
db-dtypes==1.1.1
dbus-python==1.2.16
debugpy==1.6.6
decorator==4.4.2
defusedxml==0.7.1
distributed==2022.12.1
dlib==19.24.1
dm-tree==0.1.8
docutils==0.16
dopamine-rl==4.0.6
earthengine-api==0.1.348
easydict==1.10
ecos==2.0.12
editdistance==0.6.2
en-core-web-sm @ https://github.com/explosion/spacy-
models/releases/download/en_core_web_sm-3.5.0/en_core_web_sm-3.5.0-py3-none-
any.whl
entrypoints==0.4
ephem==4.1.4
et-xmlfile==1.1.0
etils==1.2.0
etuples==0.3.8
exceptiongroup==1.1.1
fastai==2.7.12
fastcore==1.5.29
fastdownload==0.0.7
fastjsonschema==2.16.3
fastprogress==1.0.3
fastrlock==0.8.1
filelock==3.11.0
firebase-admin==5.3.0
Flask==2.2.3
flatbuffers==23.3.3
flax==0.6.8
folium==0.14.0
```

```
fonttools==4.39.3
frozendict==2.3.7
fsspec==2023.4.0
future==0.18.3
gast==0.4.0
GDAL==3.3.2
gdown==4.6.6
gensim==4.3.1
geographiclib==2.0
geopy==2.3.0
gin-config==0.5.0
glob2==0.7
google==2.0.3
google-api-core==2.11.0
google-api-python-client==2.84.0
google-auth==2.17.2
google-auth-httplib2==0.1.0
google-auth-oauthlib==1.0.0
google-cloud-bigquery==3.9.0
google-cloud-bigquery-storage==2.19.1
google-cloud-core==2.3.2
google-cloud-datastore==2.15.1
google-cloud-firestore==2.11.0
google-cloud-language==2.9.1
google-cloud-storage==2.8.0
google-cloud-translate==3.11.1
google-colab @ file:///colabtools/dist/google-colab-1.0.0.tar.gz
google-crc32c==1.5.0
google-pasta==0.2.0
google-resumable-media==2.4.1
googleapis-common-protos==1.59.0
googledrivedownloader==0.4
graphviz==0.20.1
greenlet==2.0.2
grpcio==1.53.0
grpcio-status==1.48.2
gspread==3.4.2
gspread-dataframe==3.0.8
gym==0.25.2
gym-notices==0.0.8
h5netcdf==1.1.0
h5py==3.8.0
HeapDict==1.0.1
hijri-converter==2.2.4
holidays==0.22
holoviews==1.15.4
html5lib==1.1
httpimport==1.3.0
```

```
httplib2==0.21.0
humanize==4.6.0
hyperopt==0.2.7
idna==3.4
imageio==2.25.1
imageio-ffmpeg==0.4.8
imagesize==1.4.1
imbalanced-learn==0.10.1
imblearn==0.0
imgaug==0.4.0
importlib-metadata==6.3.0
importlib-resources==5.12.0
imutils==0.5.4
inflect==6.0.4
iniconfig==2.0.0
intel-openmp==2023.1.0
ipykernel==5.5.6
ipython==7.34.0
ipython-genutils==0.2.0
ipython-sql==0.4.1
ipywidgets==7.7.1
itsdangerous==2.1.2
jax==0.4.8
jaxlib @ https://storage.googleapis.com/jax-
releases/cuda11/jaxlib-0.4.7+cuda11.cudnn86-cp39-cp39-manylinux2014_x86_64.whl
jieba==0.42.1
Jinja2==3.1.2
joblib==1.2.0
jsonpickle==3.0.1
jsonschema==4.3.3
jupyter-client==6.1.12
jupyter-console==6.1.0
jupyter-server==1.23.6
jupyter_core==5.3.0
jupyterlab-pygments==0.2.2
jupyterlab-widgets==3.0.7
kaggle==1.5.13
keras==2.12.0
keras-vis==0.4.1
kiwisolver==1.4.4
korean-lunar-calendar==0.3.1
langcodes==3.3.0
lazy_loader==0.2
libclang==16.0.0
librosa==0.10.0.post2
lightgbm==3.3.5
lit==16.0.1
llvmlite==0.39.1
```

```
locket==1.0.0
logical-unification==0.4.5
LunarCalendar==0.0.9
lxml==4.9.2
Markdown==3.4.3
markdown-it-py==2.2.0
MarkupSafe==2.1.2
matplotlib==3.7.1
matplotlib-inline==0.1.6
matplotlib-venn==0.11.9
mdurl==0.1.2
miniKanren==1.0.3
missingno==0.5.2
mistune==0.8.4
mizani==0.8.1
mkl==2019.0
ml-dtypes==0.0.4
mlxtend==0.14.0
more-itertools==9.1.0
moviepy==1.0.3
mpmath==1.3.0
msgpack==1.0.5
multipledispatch==0.6.0
multitasking==0.0.11
murmurhash==1.0.9
music21==8.1.0
natsort==8.3.1
nbclient==0.7.3
nbconvert==6.5.4
nbformat==5.8.0
nest-asyncio==1.5.6
networkx==3.1
nibabel==3.0.2
nltk==3.8.1
notebook==6.4.8
numba==0.56.4
numexpr==2.8.4
numpy==1.22.4
oauth2client==4.1.3
oauthlib==3.2.2
opencv-contrib-python==4.7.0.72
opencv-python==4.7.0.72
opencv-python-headless==4.7.0.72
openpyxl==3.0.10
opt-einsum==3.3.0
optax==0.1.4
orbax==0.1.7
osqp==0.6.2.post0
```

```
packaging==23.0
palettable==3.3.1
pandas==1.5.3
pandas-datareader==0.10.0
pandas-gbq==0.17.9
pandocfilters==1.5.0
panel==0.14.4
param==1.13.0
parso==0.8.3
partd==1.3.0
pathlib==1.0.1
pathy==0.10.1
patsy==0.5.3
pep517==0.13.0
pexpect==4.8.0
pickleshare==0.7.5
Pillow==8.4.0
pip-tools==6.6.2
platformdirs==3.2.0
plotly==5.13.1
plotnine==0.10.1
pluggy==1.0.0
pooch==1.6.0
portpicker==1.3.9
prefetch-generator==1.0.3
preshed==3.0.8
prettytable==0.7.2
proglog==0.1.10
progressbar2==4.2.0
prometheus-client==0.16.0
promise==2.3
prompt-toolkit==3.0.38
prophet==1.1.2
proto-plus==1.22.2
protobuf==3.20.3
psutil==5.9.4
psycopg2==2.9.6
ptyprocess==0.7.0
py-cpuinfo==9.0.0
py4j==0.10.9.7
pyarrow==9.0.0
pyasn1==0.4.8
pyasn1-modules==0.2.8
pycocotools==2.0.6
pycparser==2.21
pyct==0.5.0
pydantic==1.10.7
pydata-google-auth==1.7.0
```

```
pydot==1.4.2
pydot-ng==2.0.0
pydotplus==2.0.2
PyDrive==1.3.1
pyerfa==2.0.0.3
pygame==2.3.0
Pygments==2.14.0
PyGObject==3.36.0
pymc==5.1.2
PyMeeus==0.5.12
pymystem3==0.2.0
PyOpenGL==3.1.6
pyparsing==3.0.9
pyrsistent==0.19.3
PySocks==1.7.1
pytensor==2.10.1
pytest==7.2.2
python-apt==0.0.0
python-dateutil==2.8.2
python-louvain==0.16
python-slugify==8.0.1
python-utils==3.5.2
pytz==2022.7.1
pytz-deprecation-shim==0.1.0.post0
pyviz-comms==2.2.1
PyWavelets==1.4.1
PyYAML==6.0
pyzmq==23.2.1
qdldl==0.1.7
qudida==0.0.4
regex==2022.10.31
requests==2.27.1
requests-oauthlib==1.3.1
requests-unixsocket==0.2.0
rich==13.3.3
rpy2==3.5.5
rsa==4.9
scikeras==0.10.0
scikit-image==0.19.3
scikit-learn==1.2.2
scipy==1.10.1
scs==3.2.3
seaborn==0.12.2
Send2Trash==1.8.0
shapely==2.0.1
six==1.16.0
sklearn-pandas==2.2.0
smart-open==6.3.0
```

```
sniffio==1.3.0
snowballstemmer==2.2.0
sortedcontainers==2.4.0
soundfile==0.12.1
soupsieve==2.4
soxr==0.3.5
spacy==3.5.1
spacy-legacy==3.0.12
spacy-loggers==1.0.4
Sphinx==3.5.4
sphinxcontrib-applehelp==1.0.4
sphinxcontrib-devhelp==1.0.2
sphinxcontrib-htmlhelp==2.0.1
sphinxcontrib-jsmath==1.0.1
sphinxcontrib-qthelp==1.0.3
sphinxcontrib-serializinghtml==1.1.5
SQLAlchemy==2.0.9
sqlparse==0.4.3
srsly==2.4.6
statsmodels==0.13.5
sympy==1.11.1
tables==3.8.0
tabulate==0.8.10
tblib==1.7.0
tenacity==8.2.2
tensorboard==2.12.1
tensorboard-data-server==0.7.0
tensorboard-plugin-wit==1.8.1
tensorflow==2.12.0
tensorflow-datasets==4.8.3
tensorflow-estimator==2.12.0
tensorflow-gcs-config==2.12.0
tensorflow-hub==0.13.0
tensorflow-io-gcs-filesystem==0.32.0
tensorflow-metadata==1.13.0
tensorflow-probability==0.19.0
tensorstore==0.1.35
termcolor==2.2.0
terminado==0.17.1
text-unidecode==1.3
textblob==0.17.1
tf-slim==1.1.0
thinc==8.1.9
threadpoolctl==3.1.0
tifffile==2023.3.21
tinycss2==1.2.1
toml==0.10.2
tomli==2.0.1
```

```
toolz==0.12.0
torch @ https://download.pytorch.org/whl/cu118/torch-2.0.0%2Bcu118-cp39-cp39-lin
ux_x86_64.whl
torchaudio @ https://download.pytorch.org/whl/cu118/torchaudio-2.0.1%2Bcu118-cp3
9-cp39-linux_x86_64.whl
torchdata==0.6.0
torchsummary==1.5.1
torchtext==0.15.1
torchvision @ https://download.pytorch.org/whl/cu118/torchvision-0.15.1%2Bcu118-
cp39-cp39-linux_x86_64.whl
tornado==6.2
tqdm==4.65.0
traitlets==5.7.1
triton==2.0.0
tweepy==4.13.0
typer==0.7.0
typing_extensions==4.5.0
tzdata==2023.3
tzlocal==4.3
uritemplate==4.1.1
urllib3==1.26.15
vega-datasets==0.9.0
wasabi==1.1.1
wcwidth==0.2.6
webcolors==1.13
webencodings==0.5.1
websocket-client==1.5.1
Werkzeug==2.2.3
widgetsnbextension==3.6.4
wordcloud==1.8.2.2
wrapt==1.14.1
xarray==2022.12.0
xarray-einstats==0.5.1
xgboost==1.7.5
xlrd==2.0.1
yellowbrick==1.5
yfinance==0.2.17
zict==2.2.0
zipp==3.15.0
```