```
#Codefromexercise2
import pandas as pd
import numpy as np
df= pd.read_csv('penguins_size.csv')
df=df.dropna()
df['species'].replace({'Adelie': 1, 'Chinstrap': 2, 'Gentoo': 3}, inplace=True)
df['island'].replace({'Torgersen': 1, 'Biscoe': 2, 'Dream': 3}, inplace=True)
df['sex'].replace({'MALE': 1, 'FEMALE': 2}, inplace=True)
df.info()
print(df)
df.info()
df['sex'] = pd.to_numeric(df['sex'], errors='coerce')
df=df.dropna()
df.isnull().values.any()
df.head(5)
df.describe()
#Convertcategoricaltoindictorvalues
df= pd.get_dummies(df)
#Convertdatatoarray
y = np.array(df['body_mass_g'])
#Listfeatures
f_list = list(df.columns)
#Convertdatatoarray
df= np.array(df)
#Trainmodel
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(df, y, test_size = 0.2, random_state = 42)
#Estabilshbaseline
baseline_preds = test_x[:, f_list.index('flipper_length_mm')]
baseline_errors = abs(baseline_preds - test_y)
print('Average baseline error: ', round(np.mean(baseline_errors), 2))
#Random Forest
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
rf.fit(train_x, train_y);
predictions = rf.predict(test_x)
errors = abs(predictions - test_y)
#Meanabsoluteerror
print( round(np.mean(errors), 2), 'degrees.')
MAPE = 100 * (errors / test_y)
```

```
accuracy = 100 - np.mean(MAPE)
print(round(accuracy, 2), '%.')

#Build Random Hyperparameter Grid
from sklearn.model_selection import RandomizedSearchCV
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
max_df = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]
random_grid = {'n_estimators': n_estimators,
               'max_features': max_df,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)

#Random Search Training
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid,
n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
rf_random.fit(train_x, train_y)
rf_random.get_params


#Evaluate random search
def evaluate(model, test_x, test_y):
    predictions = model.predict(test_x)
    errors = abs(predictions - test_y)
    mape = 100 * np.mean(errors / test_y)
    accuracy = 100 - mape
    print('Model Performance')
    print('Average Error: {:0.4f} degrees.'.format(np.mean(errors)))
    print('Accuracy = {:0.2f}%.'.format(accuracy))

    return accuracy
base_model = RandomForestRegressor(n_estimators = 10, random_state = 42)
base_model.fit(train_x, train_y)
```

```
base_accuracy = evaluate(base_model, test_x, test_y)

param_grid = {
  'bootstrap': [True],
  'max_depth': [10, 20, 30, 40],
  'max_features': [2,3],
  'min_samples_leaf': [1, 2, 3],
  'min_samples_split': [8, 10, 12],
  'n_estimators': [1000, 1500, 2000, 2500]
}
from sklearn.model_selection import GridSearchCV
rf = RandomForestRegressor()
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                    cv = 3, n_jobs = -1, verbose = 2)
grid_search.fit(train_x, train_y)



grid_search.best_params_
best_grid = grid_search.best_estimator_
grid_accuracy = evaluate(best_grid, test_x, test_y)
print('Improvement of {:0.2f}%.'.format( 100 * (grid_accuracy - base_accuracy) /
base_accuracy))

param_grid = {
  'bootstrap': [True],
  'max_depth': [20, 30, 40, 50],
  'max_features': [3,4],
  'min_samples_leaf': [2, 3, 4],
  'min_samples_split': [10, 12, 14],
  'n_estimators': [1500, 2000, 2500, 3000]
}
from sklearn.model_selection import GridSearchCV
rf = RandomForestRegressor()
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                    cv = 3, n_jobs = -1, verbose = 2)
grid_search.fit(train_x, train_y)



grid_search.best_params_
best_grid = grid_search.best_estimator_
Import evaluate
```

```
grid_accuracy = evaluate(best_grid, test_x, test_y)
print('Improvement of {:0.2f}%.'.format( 100 * (grid_accuracy - base_accuracy) /
base_accuracy))

#Improvement of 0.67%

#Check for overfitting
from sklearn.svm import SVC
svclassifier = SVC(kernel='rbf', C=1)
svclassifier.fit(train_x, train_y)
print(svclassifier.score(train_x, train_y))
print(svclassifier.score(test_x, test_y))




#SVM
from sklearn.model_selection import train_test_split
features = pd.read_csv('penguins_size.csv')
features= pd.get_dummies(features)
features=features.dropna()
from sklearn.svm import SVC
train_set, test_set = train_test_split(features, test_size=0.2)
#Classify
x_train = train_set.iloc[:,2].values
y_train = train_set.iloc[:,2].values
x_test = test_set.iloc[:,0:2].values
y_test = test_set.iloc[:,2].values
#Time to do the thing
m= SVC()
m.fit(x_train, y_train)
print(100*m.score(x_test, y_test), '%')
from sklearn.metrics import classification_report, confusion_matrix
p=m.predict(x_test)
print(classification_report(y_test.p))
#Gridsearch
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.1, 1, 10, 100, 1000],
                        'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
```

```
                              'kernel': ['rbf']}

g = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)

g.fit(x_train, y_train)
g_predictions = g.predict(x_test)

print(classification_report(y_test, g_predictions))
#Check For Overfitting
m.score(x_train,y_train)
m.score(x_test,y_test)


#LinearRegression
x=features.drop('body_mass_g', axis=1)
y=features.body_mass_g
x_train,x_test,y_train, y_test= train_test_split(x,y, test_size=0.2, random_state=42)
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(x,y)
reg.score(x,y)
reg.get_params()

reg.set_params(**{
 'copy_X': True,
'fit_intercept': True,
 'n_jobs':0,
'positive': False})

reg.score(x,y)

reg.set_params(**{
 'copy_X': False,
'fit_intercept': True,
 'n_jobs':0,
'positive': False})

reg.score(x,y)


reg.set_params(**{
```

```
 'copy_X': False,
'fit_intercept': False,
 'n_jobs':0,
'positive': False})

reg.score(x,y)

reg.set_params(**{
 'copy_X': False,
'fit_intercept': False,
 'n_jobs':100000,
'positive': False})

reg.score(x,y)

reg.set_params(**{
 'copy_X': False,
'fit_intercept': False,
 'n_jobs':1000000,
'positive': True})

reg.score(x,y)
reg.score(x_train,y_train)
reg.score(x_test,y_test)
```