```python
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn import metrics
from sklearn.metrics import make_scorer
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
```

```python
df = pd.read_csv(r"/data/notebook_files/data.csv")
```

```python
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_m
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280

5 rows × 33 columns

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean          569 non-null    float64
 9   concave points_mean     569 non-null    float64
 10  symmetry_mean           569 non-null    float64
 11  fractal_dimension_mean  569 non-null    float64
 12  radius_se               569 non-null    float64
 13  texture_se              569 non-null    float64
```

```
print(df['diagnosis'].unique())
del df['Unnamed: 32']
del df['id']
print(df.shape)
```

```
['M' 'B']
(569, 31)
```

```
# Normalizar columna diagnosis
df['diagnosis'] = df['diagnosis'].replace(['M','B'], [1,0])
print(df['diagnosis'].unique())
print(df.shape)
```

```
[1 0]
(569, 31)
```

```


scaler = MinMaxScaler()
df_norm = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
df_norm.head()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	conca
0	1.0	0.521037	0.022658	0.545989	0.363733	0.593753	0.792037	0.703
1	1.0	0.643144	0.272574	0.615783	0.501591	0.289880	0.181768	0.203
2	1.0	0.601496	0.390260	0.595743	0.449417	0.514309	0.431017	0.462
3	1.0	0.210090	0.360839	0.233501	0.102906	0.811321	0.811361	0.565
4	1.0	0.629893	0.156578	0.630986	0.489290	0.430351	0.347893	0.463

5 rows × 31 columns

```
# Definir la etiqueta de salida y las variables predictoras
X = df_norm.iloc[:, [1, 30]].values
y = df_norm.iloc[:, 0].values
y = y.reshape(y.size, 1)
```

```
from sklearn.model_selection import train_test_split
```

```
# Definir el conjunto de entrenamiento y el conjunto de prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
# usamos 25% de los datos para la prueba
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((426, 2), (143, 2), (426, 1), (143, 1))
```

```
def tn(y_true, y_pred):
    return confusion_matrix(y_true, y_pred)[0, 0]
```

```
def fp(y_true, y_pred):
    return confusion_matrix(y_true, y_pred)[0, 1]
```

```
def fn(y_true, y_pred):
    return confusion_matrix(y_true, y_pred)[1, 0]
```

```
def tp(y_true, y_pred):
    return confusion_matrix(y_true, y_pred)[1, 1]
```

```
def accuracy(y_true, y_pred):
    cnf_matrix = confusion_matrix(y_true, y_pred)
    N = sum(map(sum, cnf_matrix))
    tp = cnf_matrix[1, 1]
    tn = cnf_matrix[0, 0]
    return round((tp + tn) / N, 2)
```

```
def acc(y_true, y_pred):
    return accuracy(y_true, y_pred)
```

```
# Propósito de la validación cruzada
scoring = {'accuracy': make_scorer(metrics.accuracy_score), 'prec': 'precision'}
scoring = {'tp': make_scorer(tp), 'tn': make_scorer(tn),
           'fp': make_scorer(fp), 'fn': make_scorer(fn),
           'acc': make_scorer(acc)}
```

```
def print_result(result):
    print("True Positive: ", result['test_tp'])
    print("True Negative: ", result['test_tn'])
    print("False Negative: ", result['test_fn'])
    print("False Positive: ", result['test_fp'])
    print("Accuracy: ", result['test_acc'])
```

```
# Lista alamcenada de las acc y rcc de las salidas de cada modelo
acc = []
roc = []
```

```
# Modelo - Random Forest
model = RandomForestClassifier(n_estimators=20, max_depth=10)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_pred_train = model.predict(X_train)
```

```
<ipython-input-20-fb2d43f20f21>:3: DataConversionWarning: A column-vector y was p
  model.fit(X_train, y_train)
```

```
ac = accuracy_score(y_test, y_pred)
acc.append(ac)
ac_train = accuracy_score(y_train, y_pred_train)
rc = roc_auc_score(y_test, y_pred)
roc.append(rc)
print("****************************************************")
print("Random Forest : ")
print ("Accuracy:", accuracy_score(y_test, y_pred))
print ("F1 score:", f1_score(y_test, y_pred))
print ("Recall:", recall_score(y_test, y_pred))
print ("Precision:", precision_score(y_test, y_pred))
print ("\n confussion matrix:\n",confusion_matrix(y_test, y_pred))
```

```
****************************************************
Random Forest :
Accuracy: 0.9230769230769231
F1 score: 0.8990825688073394
Recall: 0.8909090909090909
```

```
Precision: 0.9074074074074074

 confussion matrix:
 [[83  5]
 [ 6 49]]
```