Import libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Read the Dataset

In [3]:

```
df=pd.read_csv('heart.csv')
```

In [5]:

```
df
```

Out[5]:

	Unnamed: 0	age	sex	cp	trestbps	chol	fbs	restecg	thalach	e
0	0	0.479167	1.0	0.000000	0.292453	0.196347	0.0	0.5	0.740458	
1	1	0.500000	1.0	0.000000	0.433962	0.175799	1.0	0.0	0.641221	
2	2	0.854167	1.0	0.000000	0.481132	0.109589	0.0	0.5	0.412214	
3	3	0.666667	1.0	0.000000	0.509434	0.175799	0.0	0.5	0.687023	
4	4	0.687500	0.0	0.000000	0.415094	0.383562	1.0	0.5	0.267176	
...	...	...	...	...	...	...	...	...	...	
1020	1020	0.625000	1.0	0.333333	0.433962	0.216895	0.0	0.5	0.709924	
1021	1021	0.645833	1.0	0.000000	0.292453	0.301370	0.0	0.0	0.534351	
1022	1022	0.375000	1.0	0.000000	0.150943	0.340183	0.0	0.0	0.358779	
1023	1023	0.437500	0.0	0.000000	0.150943	0.292237	0.0	0.0	0.671756	
1024	1024	0.520833	1.0	0.000000	0.245283	0.141553	0.0	0.5	0.320611	

1025 rows × 15 columns

In [6]:

```
df.shape
```

Out[6]:

```
(1025, 15)
```

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  1025 non-null   int64  
 1   age         1025 non-null   float64
 2   sex         1025 non-null   float64
 3   cp          1025 non-null   float64
 4   trestbps    1025 non-null   float64
 5   chol        1025 non-null   float64
 6   fbs         1025 non-null   float64
 7   restecg     1025 non-null   float64
 8   thalach     1025 non-null   float64
 9   exang       1025 non-null   float64
 10  oldpeak     1025 non-null   float64
 11  slope       1025 non-null   float64
 12  ca          1025 non-null   float64
 13  thal        1025 non-null   float64
 14  target      1025 non-null   float64
dtypes: float64(14), int64(1)
memory usage: 120.2 KB
```

In [8]:

```
df.columns
```

Out[8]:

```
Index(['Unnamed: 0', 'age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg',
       'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

dropping unwanted columns

In [10]:

```
df=df.drop(columns=['ca','fbs','restecg','Unnamed: 0'],axis=1)
```

In [11]:

```
df.columns
```

Out[11]:

```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'thalach', 'exang', 'oldpeak',
       'slope', 'thal', 'target'],
      dtype='object')
```

In [12]:

```
df.describe()
```

Out[12]:

	age	sex	cp	trestbps	chol	thalach	
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1
mean	0.529878	0.695610	0.314146	0.354827	0.273973	0.596291	
std	0.189006	0.460373	0.343214	0.165252	0.117791	0.175616	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.395833	0.000000	0.000000	0.245283	0.194064	0.465649	
50%	0.562500	1.000000	0.333333	0.339623	0.260274	0.618321	
75%	0.666667	1.000000	0.666667	0.433962	0.340183	0.725191	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

In [13]:

```
df.duplicated().sum()
```

Out[13]:

```
723
```

In [14]:

```
df_unique = df.drop_duplicates()
```

In [15]:

```
df_unique
```

Out[15]:

	age	sex	cp	trestbps	chol	thalach	exang	oldpeak	slope	
0	0.479167	1.0	0.000000	0.292453	0.196347	0.740458	0.0	0.161290	1.0	1.0
1	0.500000	1.0	0.000000	0.433962	0.175799	0.641221	1.0	0.500000	0.0	1.0
2	0.854167	1.0	0.000000	0.481132	0.109589	0.412214	1.0	0.419355	0.0	1.0
3	0.666667	1.0	0.000000	0.509434	0.175799	0.687023	0.0	0.000000	1.0	1.0
4	0.687500	0.0	0.000000	0.415094	0.383562	0.267176	0.0	0.306452	0.5	0.6
...	...	...	...	...	...	...	...	...	...	
723	0.812500	0.0	0.666667	0.245283	0.194064	0.335878	0.0	0.241935	0.5	0.6
733	0.312500	0.0	0.666667	0.132075	0.034247	0.793893	0.0	0.096774	0.5	0.6
739	0.479167	1.0	0.000000	0.320755	0.294521	0.687023	1.0	0.000000	1.0	1.0
843	0.625000	1.0	1.000000	0.622642	0.335616	0.412214	0.0	0.000000	1.0	0.6
878	0.520833	1.0	0.000000	0.245283	0.141553	0.320611	0.0	0.225806	0.5	1.0

302 rows × 11 columns

In [16]:

```
df.isnull().sum()
```

Out[16]:

```
age         0
sex         0
cp          0
trestbps    0
chol        0
thalach     0
exang       0
oldpeak     0
slope       0
thal        0
target      0
dtype: int64
```

MinMax Normalization

In [17]:

```
from sklearn.preprocessing import MinMaxScaler
```

In [18]:

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

Out[18]:

	age	sex	cp	trestbps	chol	thalach	exang	oldpeak	slope	
0	0.479167	1.0	0.000000	0.292453	0.196347	0.740458	0.0	0.161290	1.0	1.
1	0.500000	1.0	0.000000	0.433962	0.175799	0.641221	1.0	0.500000	0.0	1.
2	0.854167	1.0	0.000000	0.481132	0.109589	0.412214	1.0	0.419355	0.0	1.
3	0.666667	1.0	0.000000	0.509434	0.175799	0.687023	0.0	0.000000	1.0	1.
4	0.687500	0.0	0.000000	0.415094	0.383562	0.267176	0.0	0.306452	0.5	0.
...	...	...	...	...	...	...	...	...	...	
1020	0.625000	1.0	0.333333	0.433962	0.216895	0.709924	1.0	0.000000	1.0	0.
1021	0.645833	1.0	0.000000	0.292453	0.301370	0.534351	1.0	0.451613	0.5	1.
1022	0.375000	1.0	0.000000	0.150943	0.340183	0.358779	1.0	0.161290	0.5	0.
1023	0.437500	0.0	0.000000	0.150943	0.292237	0.671756	0.0	0.000000	1.0	0.
1024	0.520833	1.0	0.000000	0.245283	0.141553	0.320611	0.0	0.225806	0.5	1.

1025 rows × 11 columns

Statistical feature extraction

In [19]:

```
features = df.drop('target', axis=1)
target_variables = df['target']

# Statistical feature extraction (row-wise)
stat_features = pd.DataFrame()
stat_features['mean'] = features.mean(axis=1)
stat_features['std'] = features.std(axis=1)
```

```
stat_features['min'] = features.min(axis=1)
stat_features['max'] = features.max(axis=1)
stat_features['range'] = features.max(axis=1) - features.min(axis=1)
stat_features['median'] = features.median(axis=1)

# Quantiles (25th and 75th)
stat_features['25%'] = features.quantile(0.25, axis=1)
stat_features['75%'] = features.quantile(0.75, axis=1)

# Variance
stat_features['variance'] = features.var(axis=1)

stat_features
```

Out[19]:

	mean	std	min	max	range	median	25%	75%	variance
0	0.486971	0.415874	0.0	1.0000	1.0000	0.385810	0.170054	0.935115	0.172951
1	0.525098	0.390599	0.0	1.0000	1.0000	0.500000	0.240340	0.910305	0.152568
2	0.527646	0.412954	0.0	1.0000	1.0000	0.450243	0.185245	0.963542	0.170531
3	0.503892	0.430478	0.0	1.0000	1.0000	0.588050	0.043950	0.921756	0.185311
4	0.322645	0.260517	0.0	0.6875	0.6875	0.345007	0.066794	0.478774	0.067869
...	...	...	...	...	...	...	...	...	...
1020	0.598578	0.350013	0.0	1.0000	1.0000	0.645833	0.358491	0.927481	0.122509
1021	0.572562	0.342323	0.0	1.0000	1.0000	0.517176	0.338931	0.911458	0.117185
1022	0.455286	0.342452	0.0	1.0000	1.0000	0.366889	0.206013	0.625000	0.117273
1023	0.321910	0.358861	0.0	1.0000	1.0000	0.221590	0.000000	0.609375	0.128781
1024	0.395409	0.363894	0.0	1.0000	1.0000	0.282947	0.162616	0.515625	0.132419

1025 rows × 9 columns

Concatenate normalized data and statistical features

In [20]:

```
# Concatenate normalized data and statistical features
combined_output = pd.concat([df_scaled, stat_features], axis=1)

print(combined_output.head())
```

```
        age  sex   cp  trestbps      chol   thalach  exang   oldpeak  slope  \
0  0.479167  1.0  0.0  0.292453  0.196347  0.740458    0.0  0.161290    1.0   
1  0.500000  1.0  0.0  0.433962  0.175799  0.641221    1.0  0.500000    0.0   
2  0.854167  1.0  0.0  0.481132  0.109589  0.412214    1.0  0.419355    0.0   
3  0.666667  1.0  0.0  0.509434  0.175799  0.687023    0.0  0.000000    1.0   
4  0.687500  0.0  0.0  0.415094  0.383562  0.267176    0.0  0.306452    0.5   

       thal  target      mean       std  min     max   range    median  \
0  1.000000     0.0  0.486971  0.415874  0.0  1.0000  1.0000  0.385810   
1  1.000000     0.0  0.525098  0.390599  0.0  1.0000  1.0000  0.500000   
2  1.000000     0.0  0.527646  0.412954  0.0  1.0000  1.0000  0.450243   
3  1.000000     0.0  0.503892  0.430478  0.0  1.0000  1.0000  0.588050   
4  0.666667     0.0  0.322645  0.260517  0.0  0.6875  0.6875  0.345007   

        25%       75%  variance  
0  0.170054  0.935115  0.172951  
1  0.240340  0.910305  0.152568  
2  0.185245  0.963542  0.170531  
3  0.043950  0.921756  0.185311  
4  0.066794  0.478774  0.067869  
```

spliting Training and testing data

In [21]:
```
from sklearn.model_selection import train_test_split
X=combined_output.drop('target',axis=1)
y=combined_output['target']

# First split
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.2, ran

# Second split
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.3, ran
```

In [22]:
```
from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (accuracy_score, precision_score, recall_score, conf
```

In [23]:
```
models1=[('RandomForest',RandomForestClassifier()),
         ('LogisticRegression',LogisticRegression()),
        ('DecisionTree',DecisionTreeClassifier())]
```

In [24]:
```
models2=[('RandomForest',RandomForestClassifier()),
         ('LogisticRegression',LogisticRegression()),
        ('DecisionTree',DecisionTreeClassifier())]
```

Define functions for additional metrics

In [25]:
```
# Define functions for additional metrics
def specificity_score(conf_matrix):
    tn, fp, fn, tp = conf_matrix.ravel()
    return tn / (tn + fp)

def npv_score(conf_matrix):
    tn, fp, fn, tp = conf_matrix.ravel()
    return tn / (tn + fn)

def fpr_score(conf_matrix):
    tn, fp, fn, tp = conf_matrix.ravel()
    return fp / (fp + tn)

def fnr_score(conf_matrix):
    tn, fp, fn, tp = conf_matrix.ravel()
    return fn / (fn + tp)

def fmeasure_score(precision, recall):
    return 2 * (precision * recall) / (precision + recall)

# Function to evaluate models
def evaluate_models(models, X_train, X_test, y_train, y_test):
    for name, model in models:
        print(f"{name}")
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        conf_matrix = confusion_matrix(y_test, y_pred)

        print(conf_matrix)
        print("Accuracy:", accuracy_score(y_test, y_pred))
        print("Precision:", precision_score(y_test, y_pred))
        print("Recall:", recall_score(y_test, y_pred))
        print("Fmeasure:", fmeasure_score(precision_score(y_test, y_pred), recal
        print("Sensitivity:", recall_score(y_test, y_pred))
        print("Specificity:", specificity_score(conf_matrix))
        print("MCC:", matthews_corrcoef(y_test, y_pred))
        print("NPV:", npv_score(conf_matrix))
        print("FPR:", fpr_score(conf_matrix))
        print("FNR:", fnr_score(conf_matrix))
        print('\n')

# Evaluate models on both splits
print("First split(80-20):")
evaluate_models(models1, X_train1, X_test1, y_train1, y_test1)

print(" Second split(70-30):")
evaluate_models(models2, X_train2, X_test2, y_train2, y_test2)
```

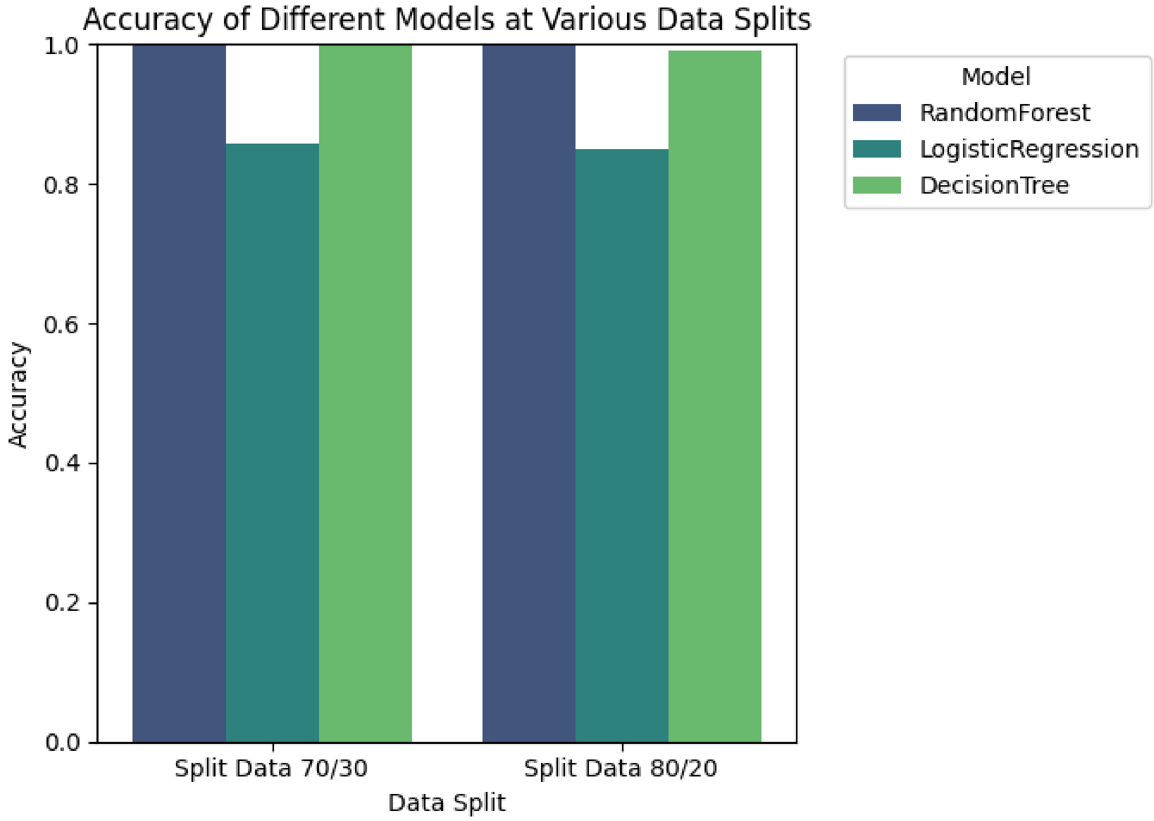
```
First split(80-20):
RandomForest
[[ 98   0]
 [  0 107]]
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
Fmeasure: 1.0
Sensitivity: 1.0
Specificity: 1.0
MCC: 1.0
NPV: 1.0
FPR: 0.0
FNR: 0.0


LogisticRegression
[[81 17]
 [12 95]]
Accuracy: 0.8585365853658536
Precision: 0.8482142857142857
Recall: 0.8878504672897196
Fmeasure: 0.8675799086757991
Sensitivity: 0.8878504672897196
Specificity: 0.826530612244898
MCC: 0.7167775340329366
NPV: 0.8709677419354839
FPR: 0.17346938775510204
FNR: 0.11214953271028037


DecisionTree
[[ 98   0]
 [  0 107]]
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
Fmeasure: 1.0
Sensitivity: 1.0
Specificity: 1.0
MCC: 1.0
NPV: 1.0
FPR: 0.0
FNR: 0.0


 Second split(70-30):
RandomForest
[[145   0]
 [  6 157]]
Accuracy: 0.9805194805194806
Precision: 1.0
Recall: 0.9631901840490797
Fmeasure: 0.98125
Sensitivity: 0.9631901840490797
Specificity: 1.0
MCC: 0.9617264301269595
NPV: 0.9602649006622517
FPR: 0.0
FNR: 0.03680981595092025
```

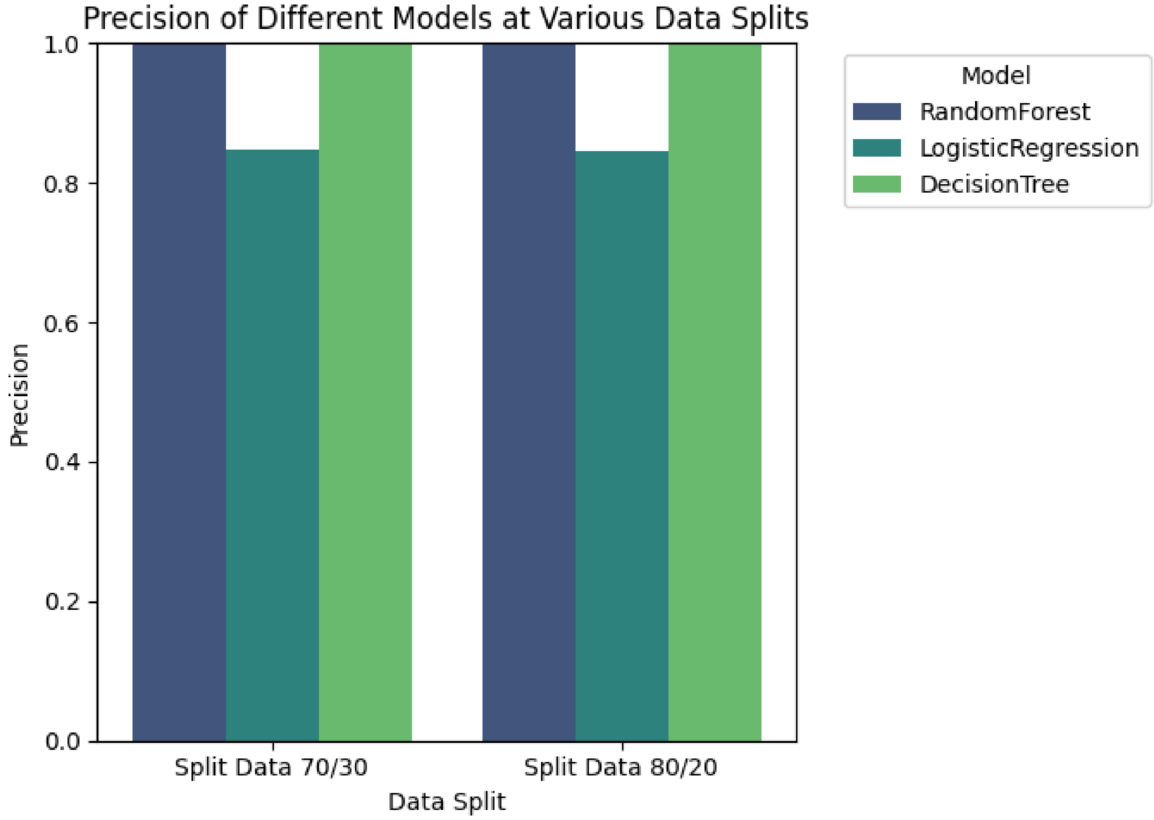
```
LogisticRegression
[[119  26]
 [ 20 143]]
Accuracy: 0.8506493506493507
Precision: 0.8461538461538461
Recall: 0.8773006134969326
Fmeasure: 0.8614457831325302
Sensitivity: 0.8773006134969326
Specificity: 0.8206896551724138
MCC: 0.700126342832296
NPV: 0.8561151079136691
FPR: 0.1793103448275862
FNR: 0.12269938650306748


DecisionTree
[[145   0]
 [  6 157]]
Accuracy: 0.9805194805194806
Precision: 1.0
Recall: 0.9631901840490797
Fmeasure: 0.98125
Sensitivity: 0.9631901840490797
Specificity: 1.0
MCC: 0.9617264301269595
NPV: 0.9602649006622517
FPR: 0.0
FNR: 0.03680981595092025
```

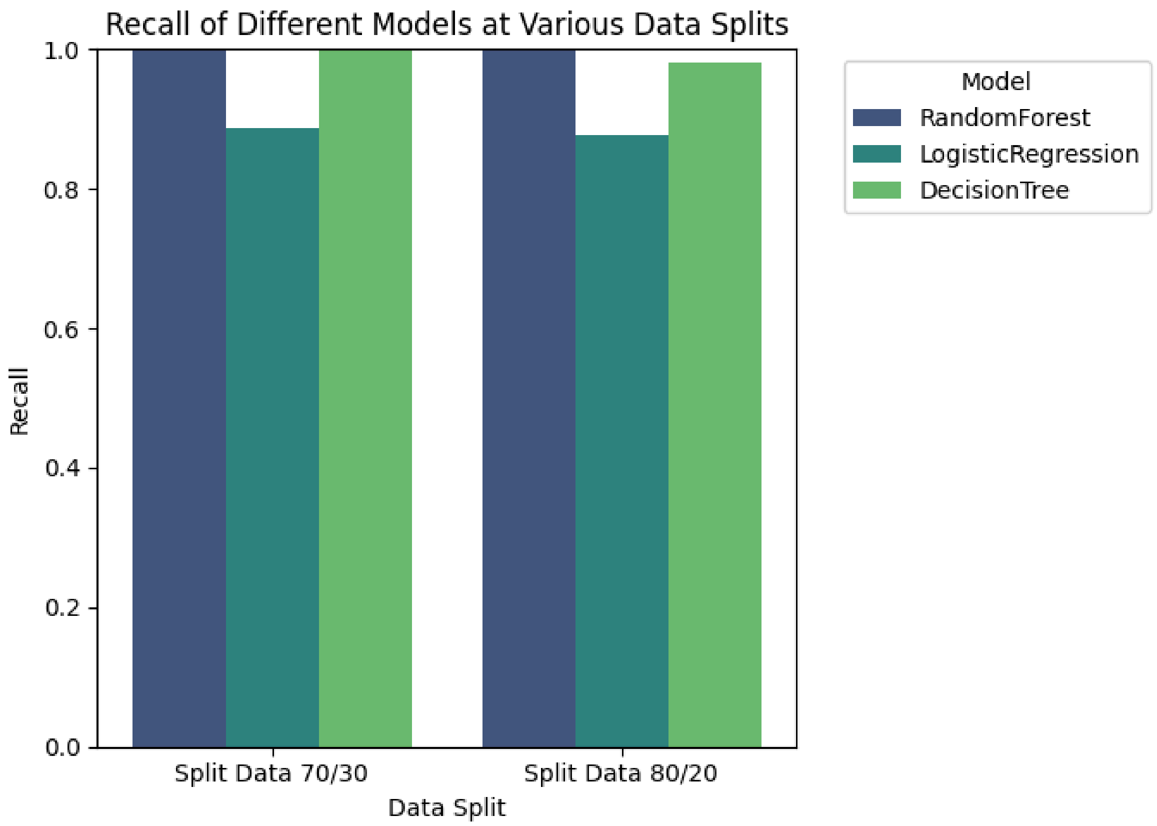
Function to collect all metrics

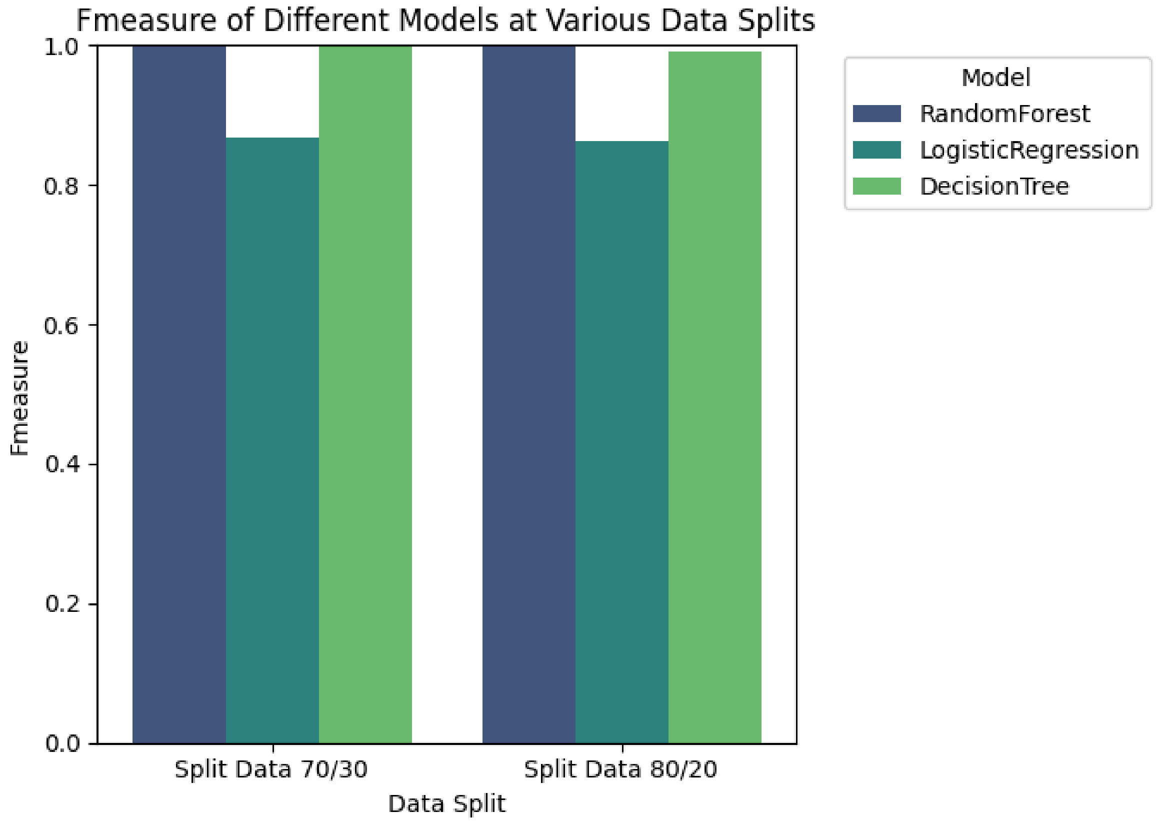
In [26]:

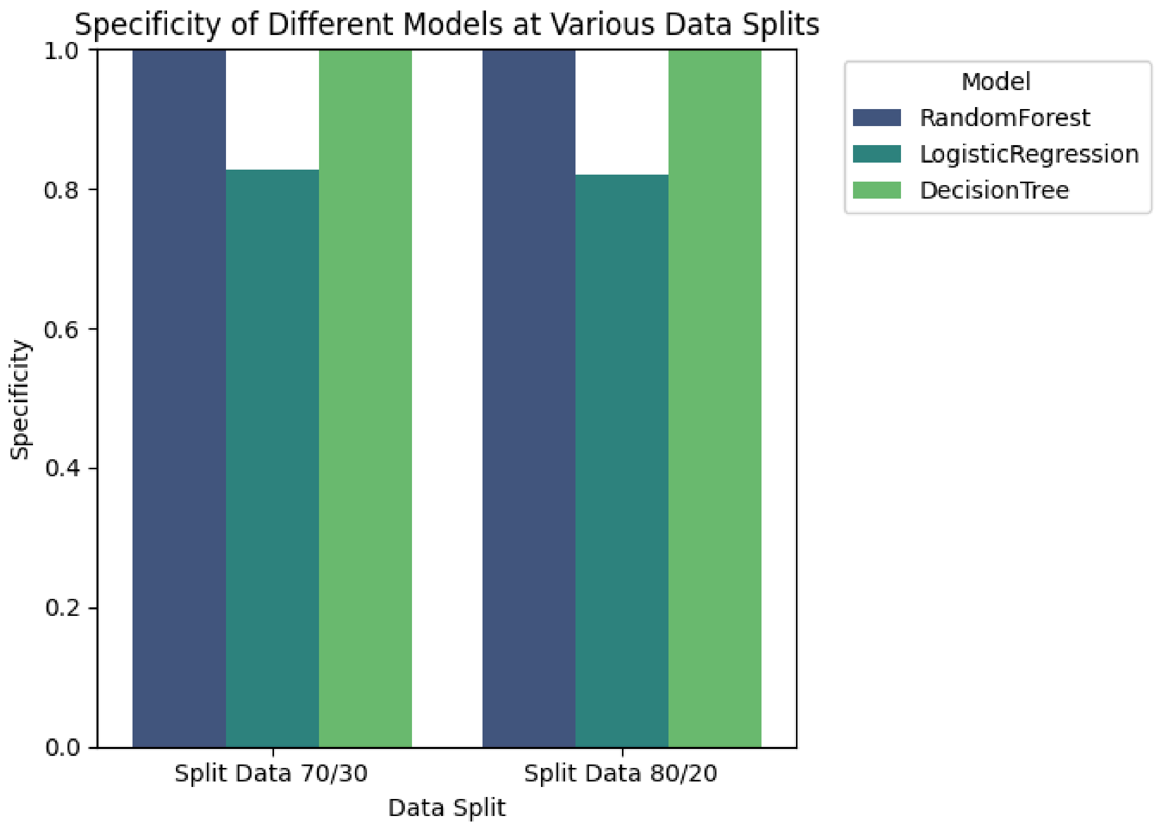
```python
# Function to collect all metrics
def get_all_metrics(models, X_train, X_test, y_train, y_test, split_name):
    metrics = []
    for name, model in models:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        conf_matrix = confusion_matrix(y_test, y_pred)
        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred)
        recall = recall_score(y_test, y_pred)
        fmeasure = fmeasure_score(precision, recall)
        specificity = specificity_score(conf_matrix)
        mcc = matthews_corrcoef(y_test, y_pred)
        npv = npv_score(conf_matrix)
        fpr = fpr_score(conf_matrix)
        fnr = fnr_score(conf_matrix)

        metrics.append({
            'Model': name,
            'Split': split_name,
            'Accuracy': accuracy,
            'Precision': precision,
            'Recall': recall,
            'Fmeasure': fmeasure,
            'Specificity': specificity,
            'MCC': mcc,
            'NPV': npv,
            'FPR': fpr,
            'FNR': fnr
        })
    return metrics

# Example model definitions
models1 = [ ('RandomForest', RandomForestClassifier()),('LogisticRegression',Log
models2 = [ ('RandomForest', RandomForestClassifier()),('LogisticRegression',Log

# Example train/test splits
X_train1, X_test1, y_train1, y_test1 = X_train1, X_test1, y_train1, y_test1
X_train2, X_test2, y_train2, y_test2 = X_train2, X_test2, y_train2, y_test2

# Collect all metrics for both splits
metrics_split1 = get_all_metrics(models1, X_train1, X_test1, y_train1, y_test1,
metrics_split2 = get_all_metrics(models2, X_train2, X_test2, y_train2, y_test2,

# Combine results into a DataFrame
metrics_df = pd.DataFrame(metrics_split1 + metrics_split2)

# Function to plot metrics
def plot_metrics(metrics_df, metric_name):
    plt.figure(figsize=(7,5))
    sns.barplot(x='Split', y=metric_name, hue='Model', data=metrics_df, palette=
    plt.title(f'{metric_name} of Different Models at Various Data Splits')
    plt.xlabel('Data Split')
    plt.ylabel(metric_name)
    plt.ylim(0, 1)
    plt.legend(title='Model', bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.xticks(rotation=0)
    plt.tight_layout()
    plt.show()

# Plot all metrics
for metric in ['Accuracy', 'Precision', 'Recall', 'Fmeasure', 'Specificity', 'MC
    plot_metrics(metrics_df, metric)
```

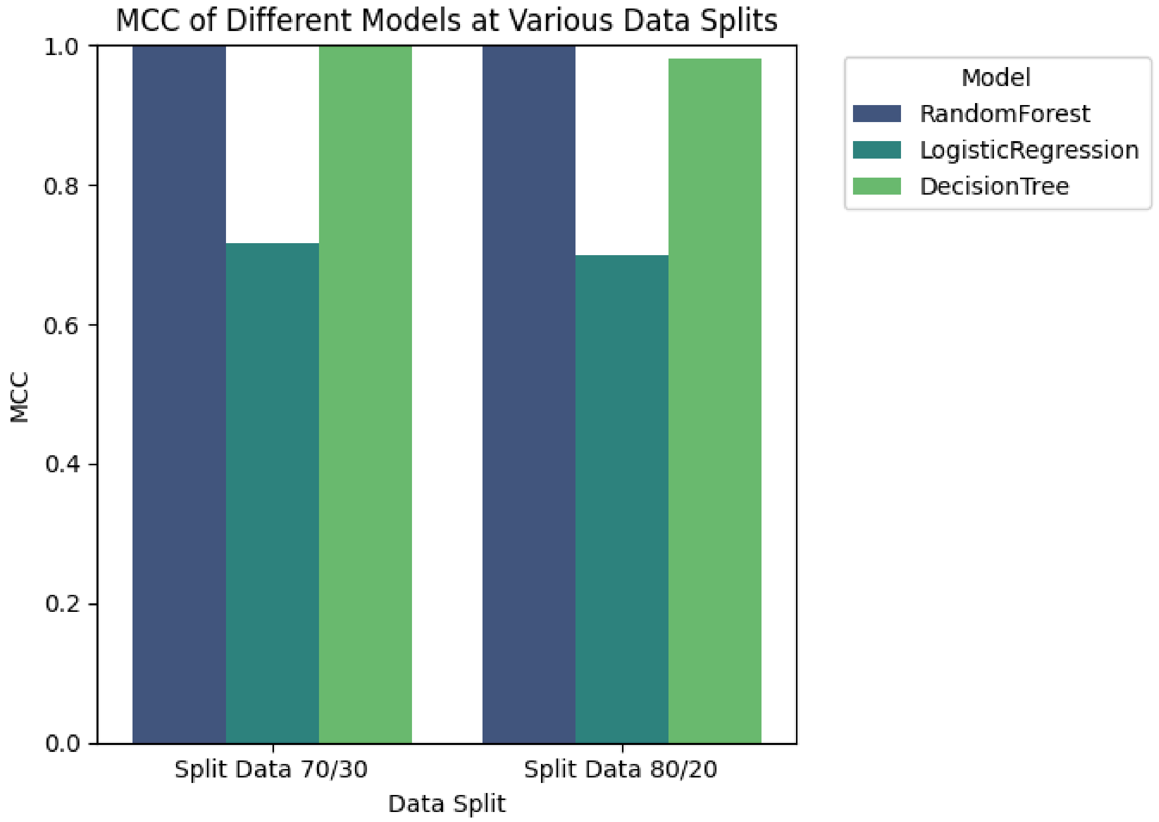

Accuracy of Different Models at Various Data Splits
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
Split Data 70/30
Split Data 80/20
Data Split
Model
RandomForest
LogisticRegression
DecisionTree

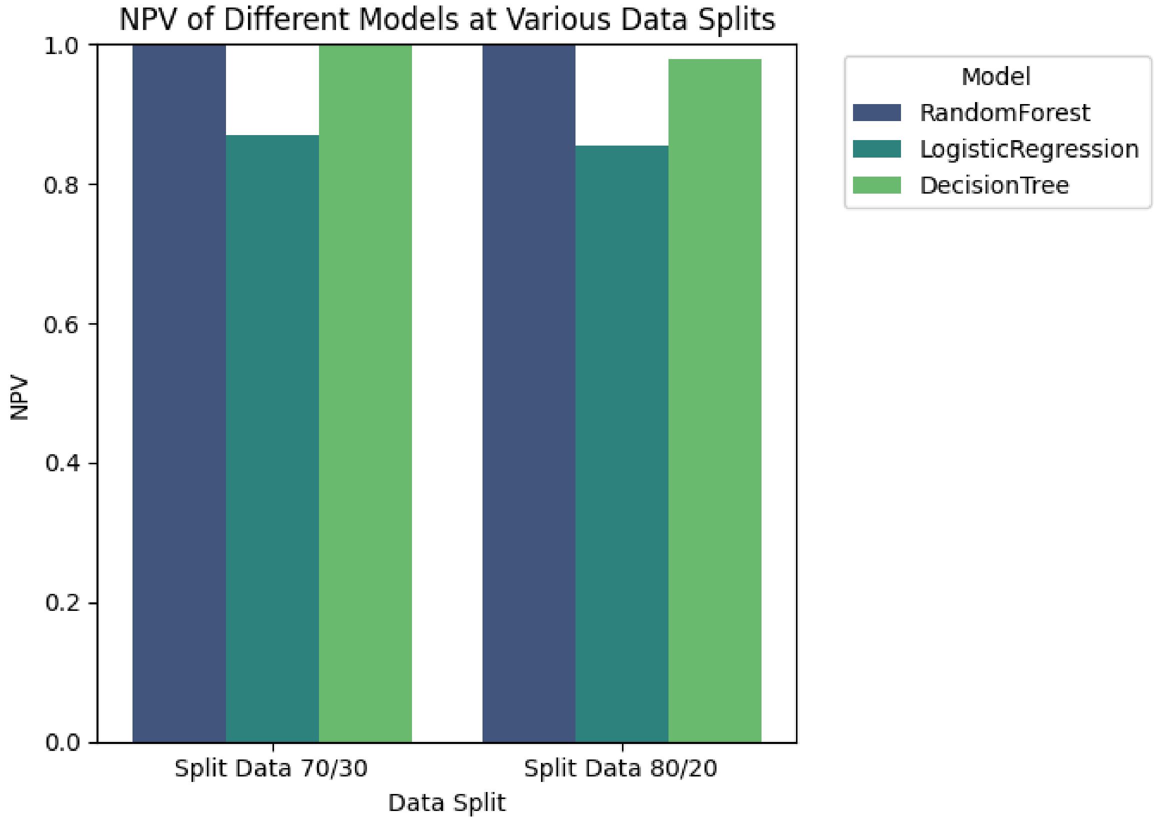

Precision of Different Models at Various Data Splits
Precision
1.0
0.8
0.6
0.4
0.2
0.0
Split Data 70/30
Split Data 80/20
Data Split
Model
RandomForest
LogisticRegression
DecisionTree

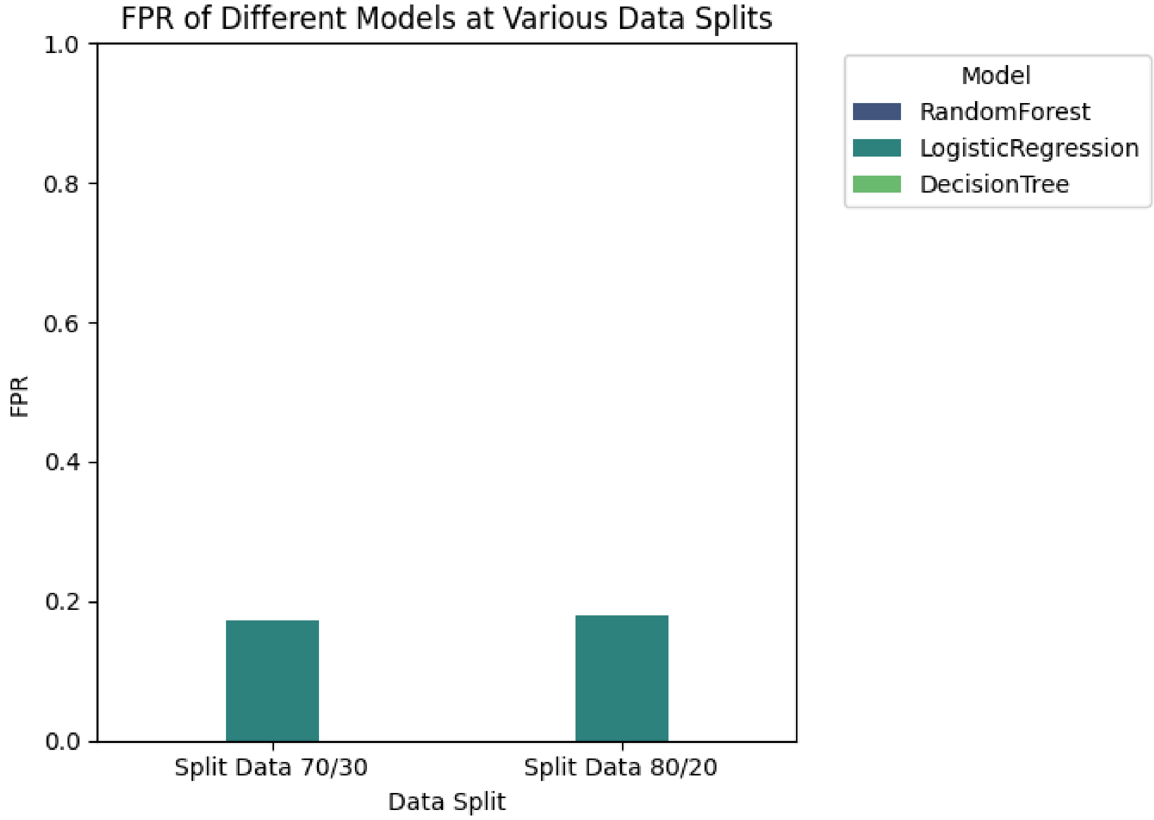

Recall of Different Models at Various Data Splits
Recall
1.0
0.8
0.6
0.4
0.2
0.0
Split Data 70/30
Split Data 80/20
Data Split
Model
RandomForest
LogisticRegression
DecisionTree

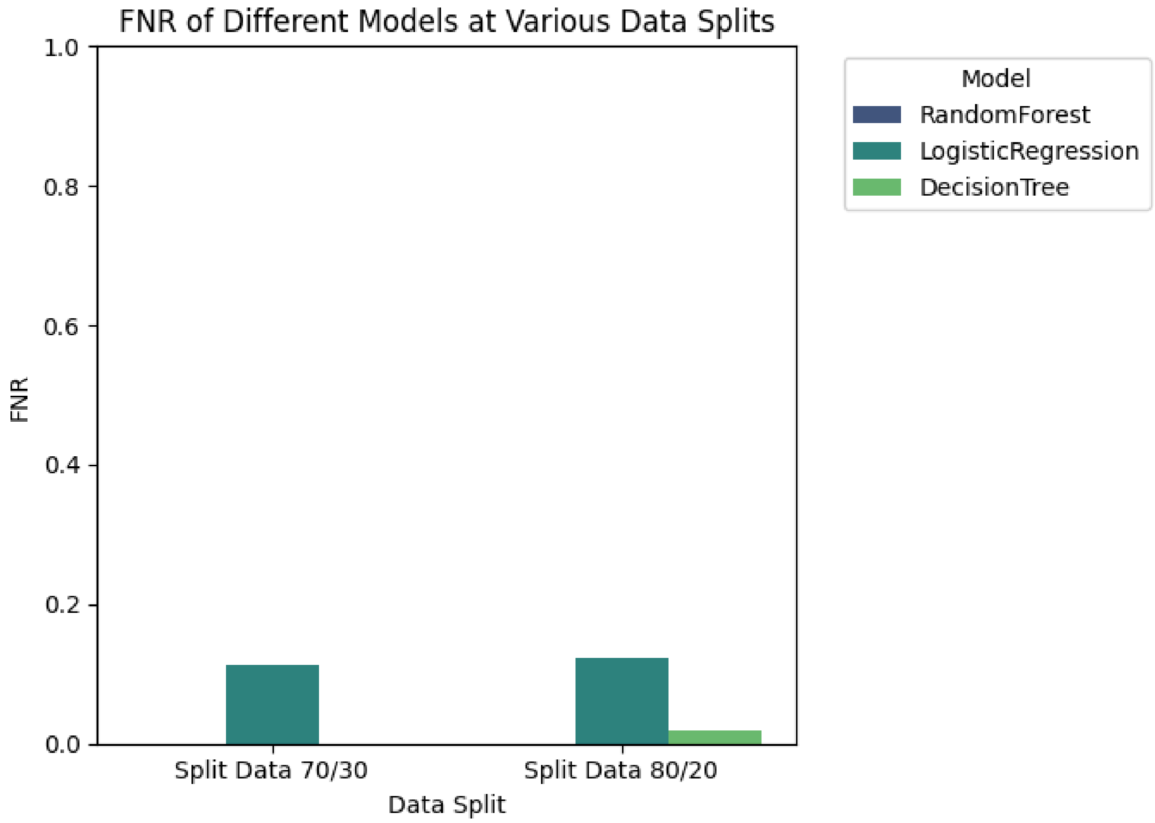

Fmeasure of Different Models at Various Data Splits
Fmeasure
1.0
0.8
0.6
0.4
0.2
0.0
Split Data 70/30
Split Data 80/20
Data Split
Model
RandomForest
LogisticRegression
DecisionTree


Specificity of Different Models at Various Data Splits
Specificity
1.0
0.8
0.6
0.4
0.2
0.0
Split Data 70/30
Split Data 80/20
Data Split
Model
RandomForest
LogisticRegression
DecisionTree


MCC of Different Models at Various Data Splits
MCC
1.0
0.8
0.6
0.4
0.2
0.0
Split Data 70/30
Split Data 80/20
Data Split
Model
RandomForest
LogisticRegression
DecisionTree


NPV of Different Models at Various Data Splits
NPV
1.0
0.8
0.6
0.4
0.2
0.0
Split Data 70/30
Split Data 80/20
Data Split
Model
RandomForest
LogisticRegression
DecisionTree


FPR of Different Models at Various Data Splits
FPR
1.0
0.8
0.6
0.4
0.2
0.0
Split Data 70/30
Split Data 80/20
Data Split
Model
RandomForest
LogisticRegression
DecisionTree


FNR of Different Models at Various Data Splits
FNR
1.0
0.8
0.6
0.4
0.2
0.0
Split Data 70/30
Split Data 80/20
Data Split
Model
RandomForest
LogisticRegression
DecisionTree

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```