In [1]:
```
pip install pdfkit
```

```
Collecting pdfkitNote: you may need to restart the kernel to use updated package
s.

  Downloading pdfkit-1.0.0-py3-none-any.whl (12 kB)
Installing collected packages: pdfkit
Successfully installed pdfkit-1.0.0
```

In [ ]:

In [ ]:

In [44]:
```
pip install nbconvert[qtpdf]
```

```
Collecting joblib>=1.1.1 (from scikit-learn)
  Downloading joblib-1.2.0-py3-none-any.whl (297 kB)
                                              0.0/298.0 kB ? eta -:--:--
     --------------                         112.6/298.0 kB 3.2 MB/s eta 0:00:01
     ------------------------------------  297.0/298.0 kB 3.1 MB/s eta 0:00:01
     -------------------------------------- 298.0/298.0 kB 3.1 MB/s eta 0:00:00
Collecting threadpoolctl>=2.0.0 (from scikit-learn)
  Downloading threadpoolctl-3.1.0-py3-none-any.whl (14 kB)
Installing collected packages: threadpoolctl, scipy, joblib, scikit-learn
Successfully installed joblib-1.2.0 scikit-learn-1.2.2 scipy-1.10.1 threadpoolctl
-3.1.0
Note: you may need to restart the kernel to use updated packages.
```

In [18]:
```
pip install imblearn
```

```
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Collecting imbalanced-learn (from imblearn)
  Downloading imbalanced_learn-0.10.1-py3-none-any.whl (226 kB)
                                              0.0/226.0 kB ? eta -:--:--
     -----                                   30.7/226.0 kB 1.3 MB/s eta 0:00:01
     ----------                              61.4/226.0 kB 1.1 MB/s eta 0:00:01
     -------------------------              153.6/226.0 kB 1.3 MB/s eta 0:00:01
     ----------------------------------     204.8/226.0 kB 1.4 MB/s eta 0:00:01
     -------------------------------------- 226.0/226.0 kB 1.3 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17.3 in c:\users\tayyar\appdata\local\pro
grams\python\python311\lib\site-packages (from imbalanced-learn->imblearn) (1.24.
3)
Requirement already satisfied: scipy>=1.3.2 in c:\users\tayyar\appdata\local\prog
rams\python\python311\lib\site-packages (from imbalanced-learn->imblearn) (1.10.
1)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\tayyar\appdata\loc
al\programs\python\python311\lib\site-packages (from imbalanced-learn->imblearn)
(1.2.2)
Requirement already satisfied: joblib>=1.1.1 in c:\users\tayyar\appdata\local\pro
grams\python\python311\lib\site-packages (from imbalanced-learn->imblearn) (1.2.
0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\tayyar\appdata\lo
cal\programs\python\python311\lib\site-packages (from imbalanced-learn->imblearn)
(3.1.0)
Installing collected packages: imbalanced-learn, imblearn
Successfully installed imbalanced-learn-0.10.1 imblearn-0.0
Note: you may need to restart the kernel to use updated packages.
```

In [13]:
```
pip install seaborn
```

```
  Cell In[13], line 1
    pip install seaborn
        ^
SyntaxError: invalid syntax
```

In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [5]:
```
from sklearn.model_selection import train_test_split, KFold, cross_val_score, Gr
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, Gradier
```

In [6]:
```
from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.pipeline import make_pipeline
```

In [7]:
```
from sklearn.metrics import recall_score, accuracy_score, precision_score, class
```

In [8]:
```
from collections import Counter
```

In [9]:
```
path = "C:/Users/Tayyar/Desktop/odev/breast_cancer.csv"
df = pd.read_csv(path)
df.head()
```

Out[9]:

	Age	Race	Marital Status	T Stage	N Stage	6th Stage	differentiate	Grade	A Stage	Tumor Size	E
0	68	White	Married	T1	N1	IIA	Poorly differentiated	3	Regional	4	
1	50	White	Married	T2	N2	IIIA	Moderately differentiated	2	Regional	35	
2	58	White	Divorced	T3	N3	IIIC	Moderately differentiated	2	Regional	63	
3	58	White	Married	T1	N1	IIA	Poorly differentiated	3	Regional	18	
4	47	White	Married	T2	N1	IIB	Poorly differentiated	3	Regional	41	

In [10]:
```
df.tail()
```

Out[10]:

	Age	Race	Marital Status	T Stage	N Stage	6th Stage	differentiate	Grade	A Stage	Tumo Siz
4019	62	Other	Married	T1	N1	IIA	Moderately differentiated	2	Regional	[illegible]
4020	56	White	Divorced	T2	N2	IIIA	Moderately differentiated	2	Regional	4
4021	68	White	Married	T2	N1	IIB	Moderately differentiated	2	Regional	2
4022	58	Black	Divorced	T2	N1	IIB	Moderately differentiated	2	Regional	4
4023	46	White	Married	T2	N1	IIB	Moderately differentiated	2	Regional	3

In [11]:
```
# Renaming some column names
df.rename({'Tumor Size' : 'Tumor Size (mm)', 'T Stage ': 'T Stage', 'Reginol Noc
# Changinge the values of 'Grade' column to uniform values
df['Grade'] = df['Grade'].map({'1' : 'Grade 1', '2' : 'Grade 2', '3' : 'Grade 3'
```

In [12]:
```
df.shape
```

Out[12]:
```
(4024, 16)
```

In [13]:
```
df.ndim
```

Out[13]:
```
2
```

In [14]:
```
df.info
```

```
Out[14]: <bound method DataFrame.info of        Age   Race Marital Status T Stage N Stage 
6th Stage  
0       68  White        Married      T1      N1       IIA  \
1       50  White        Married      T2      N2      IIIA   
2       58  White       Divorced      T3      N3      IIIC   
3       58  White        Married      T1      N1       IIA   
4       47  White        Married      T2      N1       IIB   
...    ...    ...            ...     ...     ...       ...   
4019    62  Other        Married      T1      N1       IIA   
4020    56  White       Divorced      T2      N2      IIIA   
4021    68  White        Married      T2      N1       IIB   
4022    58  Black       Divorced      T2      N1       IIB   
4023    46  White        Married      T2      N1       IIB   

                  Differentiate    Grade  A Stage  Tumor Size (mm)  
0         Poorly differentiated  Grade 3  Regional                4  \
1     Moderately differentiated  Grade 2  Regional               35   
2     Moderately differentiated  Grade 2  Regional               63   
3         Poorly differentiated  Grade 3  Regional               18   
4         Poorly differentiated  Grade 3  Regional               41   
...                         ...      ...       ...              ...   
4019  Moderately differentiated  Grade 2  Regional                9   
4020  Moderately differentiated  Grade 2  Regional               46   
4021  Moderately differentiated  Grade 2  Regional               22   
4022  Moderately differentiated  Grade 2  Regional               44   
4023  Moderately differentiated  Grade 2  Regional               30   

     Estrogen Status Progesterone Status  Regional Node Examined  
0           Positive            Positive                      24  \
1           Positive            Positive                      14   
2           Positive            Positive                      14   
3           Positive            Positive                       2   
4           Positive            Positive                       3   
...              ...                 ...                     ...   
4019        Positive            Positive                       1   
4020        Positive            Positive                      14   
4021        Positive            Negative                      11   
4022        Positive            Positive                      11   
4023        Positive            Positive                       7   

      Regional Node Positive  Survival Months Status  
0                          1               60  Alive  
1                          5               62  Alive  
2                          7               75  Alive  
3                          1               84  Alive  
4                          1               50  Alive  
...                      ...              ...    ...  
4019                       1               49  Alive  
4020                       8               69  Alive  
4021                       3               69  Alive  
4022                       1               72  Alive  
4023                       2              100  Alive  

[4024 rows x 16 columns]>
```

```
In [15]: df.notnull().sum()
```

```
Out[15]: Age                        4024
         Race                       4024
         Marital Status             4024
         T Stage                    4024
         N Stage                    4024
         6th Stage                  4024
         Differentiate              4024
         Grade                      4024
         A Stage                    4024
         Tumor Size (mm)            4024
         Estrogen Status            4024
         Progesterone Status        4024
         Regional Node Examined     4024
         Regional Node Positive     4024
         Survival Months            4024
         Status                     4024
         dtype: int64
```

```
In [16]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4024 entries, 0 to 4023
Data columns (total 16 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   Age                     4024 non-null   int64 
 1   Race                    4024 non-null   object
 2   Marital Status          4024 non-null   object
 3   T Stage                 4024 non-null   object
 4   N Stage                 4024 non-null   object
 5   6th Stage               4024 non-null   object
 6   Differentiate           4024 non-null   object
 7   Grade                   4024 non-null   object
 8   A Stage                 4024 non-null   object
 9   Tumor Size (mm)         4024 non-null   int64 
 10  Estrogen Status         4024 non-null   object
 11  Progesterone Status     4024 non-null   object
 12  Regional Node Examined  4024 non-null   int64 
 13  Regional Node Positive  4024 non-null   int64 
 14  Survival Months         4024 non-null   int64 
 15  Status                  4024 non-null   object
dtypes: int64(5), object(11)
memory usage: 503.1+ KB
```

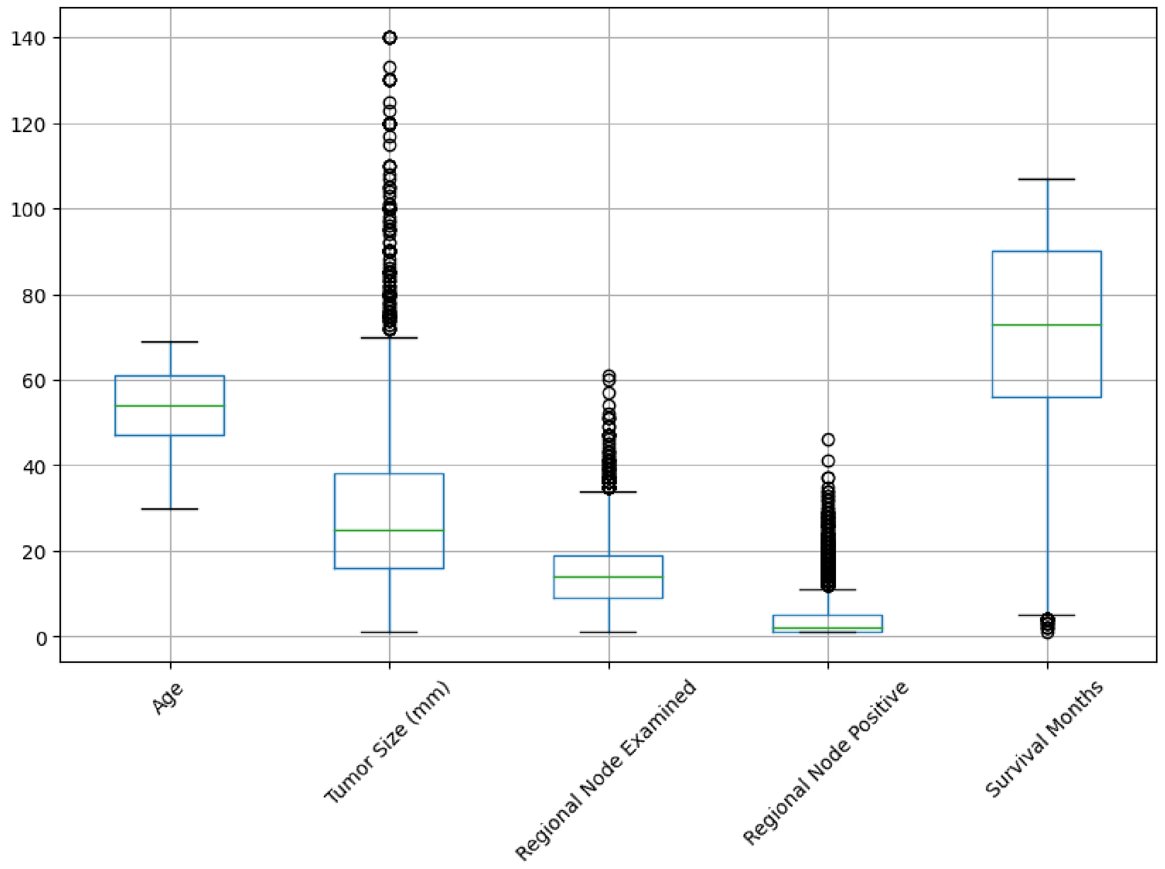
```
In [17]: df.describe()
```

Out[17]:

	Age	Tumor Size (mm)	Regional Node Examined	Regional Node Positive	Survival Months
count	4024.000000	4024.000000	4024.000000	4024.000000	4024.000000
mean	53.972167	30.473658	14.357107	4.158052	71.297962
std	8.963134	21.119696	8.099675	5.109331	22.921430
min	30.000000	1.000000	1.000000	1.000000	1.000000
25%	47.000000	16.000000	9.000000	1.000000	56.000000
50%	54.000000	25.000000	14.000000	2.000000	73.000000
75%	61.000000	38.000000	19.000000	5.000000	90.000000
max	69.000000	140.000000	61.000000	46.000000	107.000000

In [18]:
```
numeric_columns = ['Age', 'Tumor Size (mm)', 'Regional Node Examined', 'Regional
numeric_data = df[numeric_columns]
```

In [19]:
```
plt.figure(figsize=(10, 6))
numeric_data.boxplot()
plt.xticks(rotation=45)
plt.show()
```



In [20]:
```
Q1 = numeric_data.quantile(0.25)
Q3 = numeric_data.quantile(0.75)
IQR = Q3 - Q1

lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR
```

```
outliers = ((numeric_data < lower_limit) | (numeric_data > upper_limit)).any(axi
outlier_rows = numeric_data[outliers]
```

In [21]:
```
outlier_cols = ['Tumor Size (mm)', 'Regional Node Examined', 'Regional Node Posi

outlier_index = []
outlier_ind_list=[]

for col in outlier_cols:

    Q1 = np.percentile(df[col], 25, interpolation='midpoint')
    Q3 = np.percentile(df[col], 75, interpolation='midpoint')

    IQR = Q3 - Q1

    upper_bound = Q3 + (1.5 * IQR)
    lower_bound = Q1 - (1.5 * IQR)

    upper_outliers = df[col] >= upper_bound
    lower_outliers = df[col] <= lower_bound

    outliers = df[upper_outliers | lower_outliers].index

    outlier_index.extend(outliers)

# Outlier index with count
outlier_index = Counter(outlier_index)

# Storing only unique outlier index
for key in outlier_index.keys():
    outlier_ind_list.append(key)

# Dropping rows containing outliers
#df = df.drop(outlier_ind_list)

print( f'Number of outliers: {len(outlier_ind_list)}' )
```

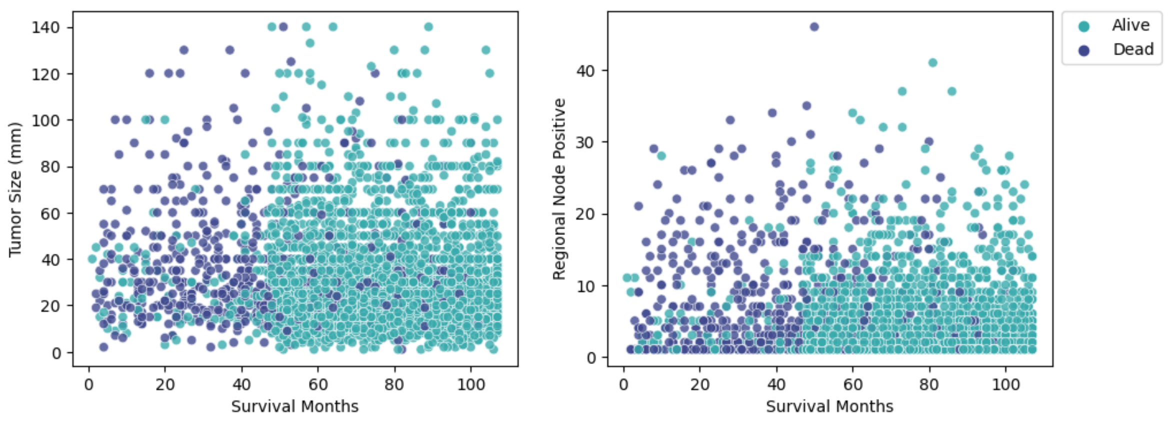
```
Number of outliers: 617
C:\Users\Tayyar\AppData\Local\Temp\ipykernel_11788\2044867857.py:8: DeprecationWa
rning: the `interpolation=` argument to percentile was renamed to `method=`, whic
h has additional options.
Users of the modes 'nearest', 'lower', 'higher', or 'midpoint' are encouraged to
review the method they used. (Deprecated NumPy 1.22)
  Q1 = np.percentile(df[col], 25, interpolation='midpoint')
C:\Users\Tayyar\AppData\Local\Temp\ipykernel_11788\2044867857.py:9: DeprecationWa
rning: the `interpolation=` argument to percentile was renamed to `method=`, whic
h has additional options.
Users of the modes 'nearest', 'lower', 'higher', or 'midpoint' are encouraged to
review the method they used. (Deprecated NumPy 1.22)
  Q3 = np.percentile(df[col], 75, interpolation='midpoint')
```

In [22]:
```
status_counts = df['Status'].value_counts()
total_patients = len(df)

for status, count in status_counts.items():
    percentage = count * 100 / total_patients
    print(f"{status}: {count} patients ({percentage:.2f}%)")
```

```
Alive: 3408 patients (84.69%)
Dead: 616 patients (15.31%)
```

In [23]:
```
# Relationship of Survival Months with Tumor Size and Regional Node Positive
scatter_cols = ['Tumor Size (mm)', 'Regional Node Positive']

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(11,4), dpi=100)
fig.suptitle('Relationship of Survival Months with Tumor Size and Regional Node
for col, ax in zip(scatter_cols, axes.ravel()):
    sns.scatterplot(data=df, x='Survival Months', y=col, hue='Status', palette='
    ax.legend_.remove()
ax.legend(loc=(1.02, 0.85));
```

Relationship of Survival Months with Tumor Size and Regional Node Positive



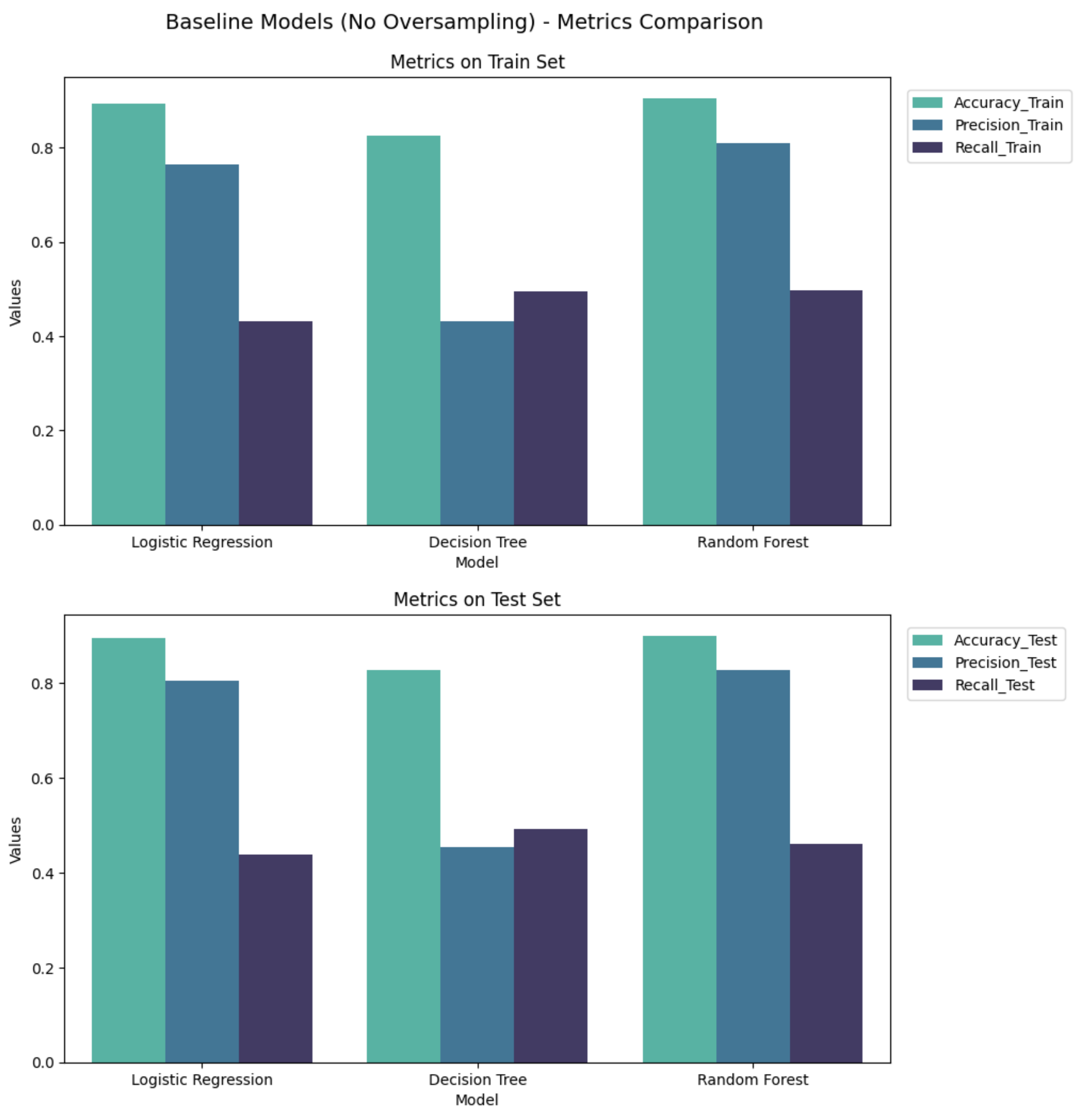
In [24]:
```
seed = 101

# For train dataset validation, choose kFold
kfold = KFold(n_splits=5, random_state=seed, shuffle=True)

# Class labels for target column
cls_labels = ['Alive', 'Dead']

# Model definition
cls_models = {
    'Logistic Regression': LogisticRegression(max_iter=100000, random_state=seed
    'Decision Tree': DecisionTreeClassifier(random_state=seed),
    'Random Forest': RandomForestClassifier(random_state=seed)
}

# Plot confusion matrix
def plot_confusion_matrix(models, y_test, y_pred_list, approach, rows, cols, w,
    fig, axes = plt.subplots(nrows=rows, ncols=cols, figsize=(w, h), dpi=100)
    fig.suptitle(f'Confusion Matrix for different models - {approach}', fontsize
    for name, y_pred, ax in zip(models, y_pred_list, axes.ravel()):
        conf_mat = confusion_matrix(y_test, y_pred)
        sns.heatmap(conf_mat, annot=True, ax=ax, fmt='d', cmap='mako_r')
        ax.set_title(name)
        ax.set_xticklabels(cls_labels)
        ax.set_yticklabels(cls_labels)
    plt.tight_layout()
    plt.show()


# Baseline models - No oversampling
def run_baseline_models():
    accuracy_list = []
    recall_list = []
    precision_list = []

    for name, model in cls_models.items():

        score_accuracy = cross_val_score(model, X_train, y_train, cv=kfold, scor
        score_precision = cross_val_score(model, X_train, y_train, cv=kfold, sco
        score_recall = cross_val_score(model, X_train, y_train, cv=kfold, scorin

        mean_accuracy_train = np.mean(score_accuracy.mean())
        mean_precision_train = np.mean(score_precision.mean())
        mean_recall_train = np.mean(score_recall.mean())

        accuracy_train.append(mean_accuracy_train)
        precision_train.append(mean_precision_train)
        recall_train.append(mean_recall_train)

        model.fit(X_train, y_train)

        # Predict target
        y_predict = model.predict(X_test)
        y_pred_baseline.append(y_predict)

        # Metrics
        accuracy_pred = accuracy_score(y_test, y_predict)
        precision_pred = precision_score(y_test, y_predict)
        recall_pred = recall_score(y_test, y_predict)

        accuracy_test.append(accuracy_pred)
        precision_test.append(precision_pred)
        recall_test.append(recall_pred)
```

In [25]:
```
# Assigning "0" to Alive class and "1" to Dead class
df['Status'] = df['Status'].map({'Alive': 0, 'Dead': 1}).astype('int64')

# Scaling the continuous feature variables with StandardScaler
scaler = StandardScaler()
num_cols = df.select_dtypes('int').columns[:-1]  # Not considering target
df[num_cols] = scaler.fit_transform(df[num_cols])

# Convert categorical string values to numeric values
X = pd.get_dummies(df.drop('Status', axis=1), drop_first=True)
y = df['Status']

# Split the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random

print( f'Shape of X_train: {X_train.shape}')
print( f'Shape of X_test: {X_test.shape}' )
```

```
Shape of X_train: (2816, 29)
Shape of X_test: (1208, 29)
```

In [26]:
```
# Fit the baseline models on train set and predict target in test set
accuracy_train = []
precision_train = []
recall_train = []

accuracy_test = []
```

```
precision_test = []
recall_test = []

y_pred_baseline = []

# Run the baseline models
run_baseline_models()

# Store the metrics in a dataframe
metrics_base_train = pd.DataFrame(list(zip(cls_models.keys(), accuracy_train,
                                           precision_train, recall_train)),
                                  columns=['Model', 'Accuracy_Train', 'Precision

metrics_base_test = pd.DataFrame(list(zip(cls_models.keys(), accuracy_test,
                                           precision_test, recall_test)),
                                  columns=['Model', 'Accuracy_Test', 'Precision_

metrics_melt_base_train = pd.melt(metrics_base_train, id_vars=['Model'],
                                  value_vars=['Accuracy_Train', 'Precision_Train

metrics_melt_base_test = pd.melt(metrics_base_test, id_vars=['Model'],
                                 value_vars=['Accuracy_Test', 'Precision_Test',

# Plot the metrics
fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, figsize=(10, 12), dpi=100)
fig.suptitle('Baseline Models (No Oversampling) - Metrics Comparison', fontsize=
sns.barplot(data=metrics_melt_base_train, x='Model', y='value', hue='variable',
sns.barplot(data=metrics_melt_base_test, x='Model', y='value', hue='variable', p

ax1.set_ylabel('Values')
ax1.set_title('Metrics on Train Set')
ax1.legend(loc=(1.02, 0.81))

ax2.set_ylabel('Values')
ax2.set_title('Metrics on Test Set')
ax2.legend(loc=(1.02, 0.81));
```



In [34]:
```
# To implement the above steps, we will use a pipeline from imbalanced-learn lib

# Function definition
def run_sampling_model(estimator, method):

    cv = kfold

    accuracy_list = []
    recall_list = []
    precision_list = []

    sampling_params = {'sampling_strategy' : 'auto', 'random_state' : seed}

    if method == 'RandomOverSample':
        rand_over = RandomOverSampler(**sampling_params)
    elif method == 'SMOTE':
        rand_over = SMOTE(**sampling_params)

    for train, val in cv.split(X_train, y_train):

        pipeline = make_pipeline(rand_over, estimator)

        model = pipeline.fit(X_train.iloc[train], y_train.iloc[train])
        y_val_predict = model.predict(X_train.iloc[val])

        accuracy_list.append(accuracy_score(y_train.iloc[val], y_val_predict))

        precision_list.append(precision_score(y_train.iloc[val], y_val_predict))

        recall_list.append(recall_score(y_train.iloc[val], y_val_predict))

    mean_accuracy_train = np.mean(accuracy_list)
    mean_precision_train = np.mean(precision_list)
    mean_recall_train = np.mean(recall_list)

    accuracy_train.append(mean_accuracy_train)
    precision_train.append(mean_precision_train)
    recall_train.append(mean_recall_train)

    # Predict target
    y_predict = model.predict(X_test)

    # Test result metrics
    accuracy_pred = accuracy_score(y_test, y_predict)
    precision_pred = precision_score(y_test, y_predict)
    recall_pred = recall_score(y_test, y_predict)

    accuracy_test.append(accuracy_pred)
    precision_test.append(precision_pred)
    recall_test.append(recall_pred)

    if method == 'RandomOverSample':
        y_pred_list_rand.append(y_predict)
    elif method == 'SMOTE':
        y_pred_list_smote.append(y_predict)
```

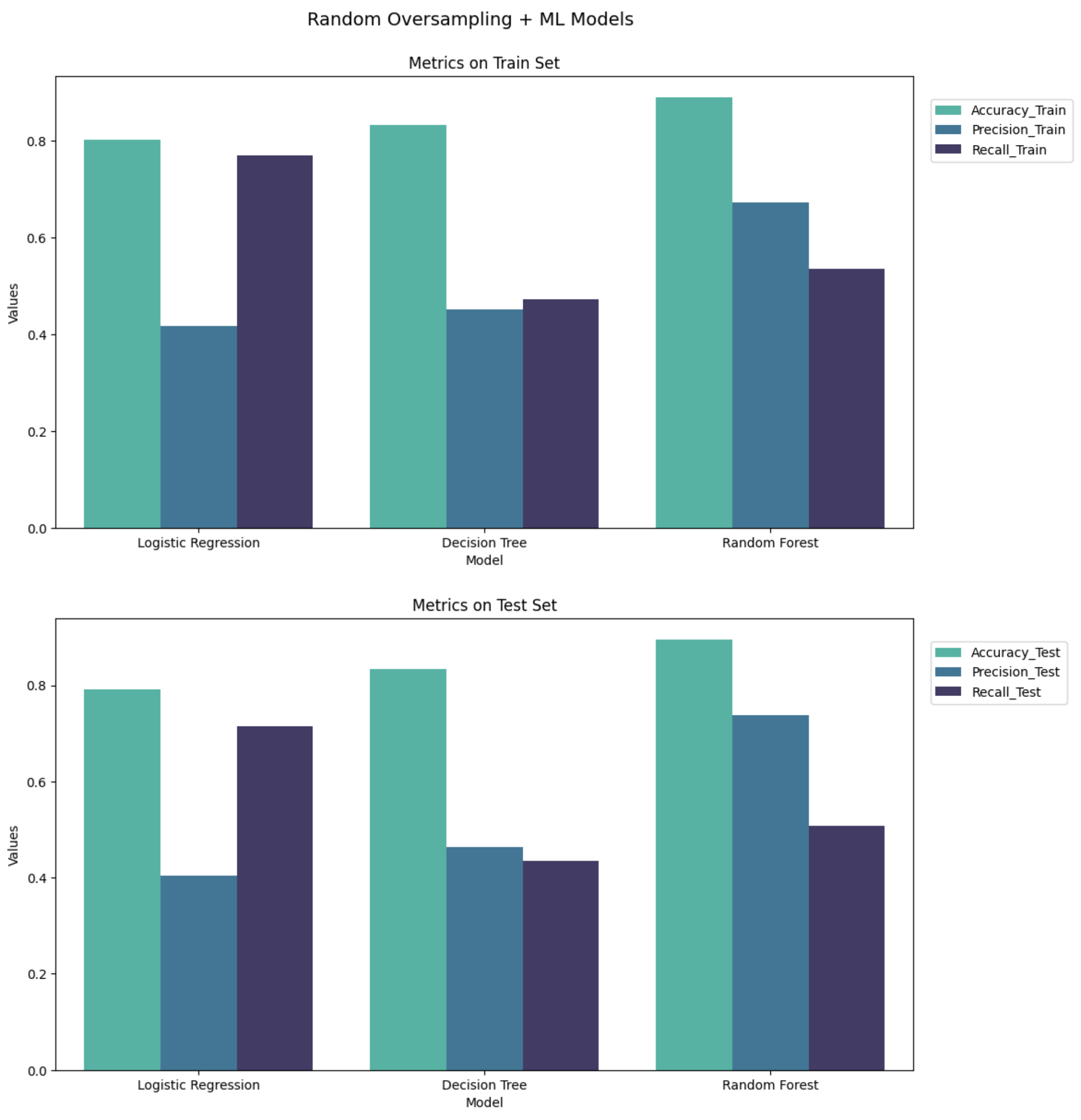
In [35]:
```
# Random Oversampling

accuracy_train = []
precision_train = []
recall_train = []

accuracy_test = []
precision_test = []
recall_test = []

y_pred_list_rand = []

method = 'RandomOverSample'

# Train model
for name, model in cls_models.items():
    run_sampling_model(model, method)

# Store metrics in a dataframe
metrics_rand_train = pd.DataFrame(list(zip(cls_models.keys(), accuracy_train,
                                           precision_train, recall_train)),
                                  columns=['Model', 'Accuracy_Train', 'Precision


metrics_rand_test = pd.DataFrame(list(zip(cls_models.keys(), accuracy_test,
                                          precision_test, recall_test)),
                                  columns=['Model', 'Accuracy_Test', 'Precision_

metrics_melt_rand_train = pd.melt(metrics_rand_train, id_vars=['Model'],
                                  value_vars=['Accuracy_Train', 'Precision_Train

metrics_melt_rand_test = pd.melt(metrics_rand_test, id_vars=['Model'],
                                 value_vars=['Accuracy_Test', 'Precision_Test',
```

In [36]:

```
# Plot metrics obtained for Train set and Test set

fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, figsize=(12, 14), dpi=100)
fig.suptitle('Random Oversampling + ML Models', fontsize=14, y=0.93)
sns.barplot(data=metrics_melt_rand_train, x='Model', y='value', hue='variable',
sns.barplot(data=metrics_melt_rand_test, x='Model', y='value', hue='variable', p

ax1.set_ylabel('Values')
ax1.set_title('Metrics on Train Set')
ax1.legend(loc=(1.02, 0.81))

ax2.set_ylabel('Values')
ax2.set_title('Metrics on Test Set')
ax2.legend(loc=(1.02, 0.81))
```

Out[36]:

```
<matplotlib.legend.Legend at 0x1434daa1490>
```



```
In [37]: logreg = LogisticRegression(max_iter=100000, random_state=seed)
         logreg.fit(X_train, y_train)

         # Test veri seti üzerinde tahmin yapma
         y_pred_logreg = logreg.predict(X_test)
```

```
In [38]: rf = RandomForestClassifier(random_state=seed)
         rf.fit(X_train, y_train)

         # Test veri seti üzerinde tahmin yapma
         y_pred_rf = rf.predict(X_test)
```

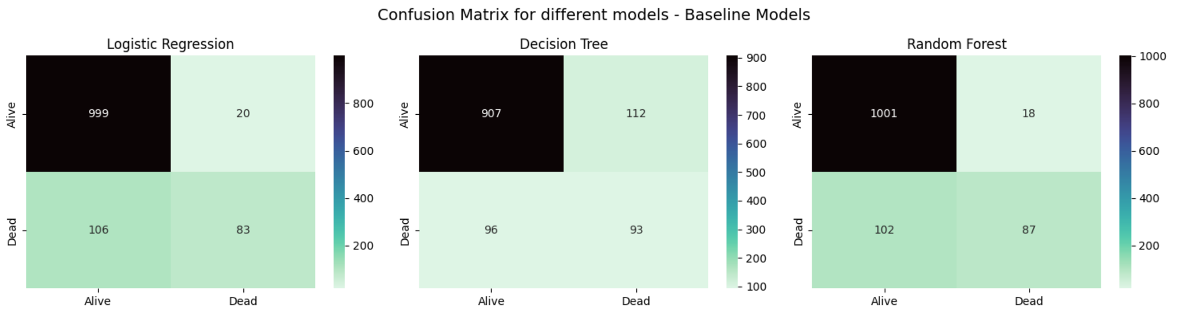
```
In [39]: dt = DecisionTreeClassifier(random_state=seed)
         dt.fit(X_train, y_train)

         # Test veri seti üzerinde tahmin yapma
         y_pred_dt = dt.predict(X_test)
```

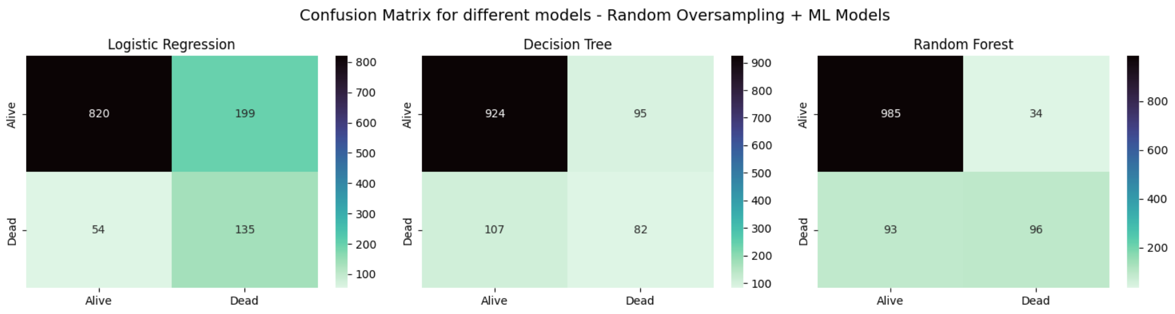
```
In [40]: # Model isimleri
         model_names = ['Logistic Regression', 'Random Forest', 'Decision Tree']

         # Tahmin sonuçları
```

```
y_preds = [y_pred_logreg, y_pred_rf, y_pred_dt]

# Değerlendirme metrikleri
metrics = ['accuracy', 'precision', 'recall']

# Her model için değerlendirme metriklerini hesaplama
for model_name, y_pred in zip(model_names, y_preds):
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)

    print(f"Model: {model_name}")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print("\n")
```

```
Model: Logistic Regression
Accuracy: 0.8957
Precision: 0.8058
Recall: 0.4392


Model: Random Forest
Accuracy: 0.9007
Precision: 0.8286
Recall: 0.4603


Model: Decision Tree
Accuracy: 0.8278
Precision: 0.4537
Recall: 0.4921
```

In [43]:
```
# Confusion Matrix for baseline models
plot_confusion_matrix(cls_models.keys(), y_test, y_pred_baseline, 'Baseline Mode

# Confusion Matrix for Random Oversampling + ML Models
plot_confusion_matrix(cls_models.keys(), y_test, y_pred_list_rand, 'Random Overs
```




Confusion Matrix for different models - Random Oversampling + ML Models
Logistic Regression
Decision Tree
Random Forest
Alive
Dead
820
199
54
135
924
95
107
82
985
34
93
96

```
In [3]:
```

```
In [1]:
```