Project Title:

Predictive Modeling for Breast Cancer Diagnosis Using Machine Learning

Feature Descriptions for Breast Cancer Dataset

- **id**: Unique identifier for each sample.
- **diagnosis**: Target variable indicating the diagnosis (**M** = Malignant, **B** = Benign).

Mean Features:

- **radius_mean**: Mean radius of the tumor cells.
- **texture_mean**: Mean texture (variation in gray levels) of the tumor cells.
- **perimeter_mean**: Mean perimeter of the tumor cells.
- **area_mean**: Mean area of the tumor cells.
- **smoothness_mean**: Mean smoothness (local variation in radius lengths) of the tumor cells.
- **compactness_mean**: Mean compactness ($perimeter^2$ / area - 1.0) of the tumor cells.
- **concavity_mean**: Mean concavity (severity of concave portions of the contour) of the tumor cells.
- **concave points_mean**: Mean number of concave portions of the tumor cell contours.
- **symmetry_mean**: Mean symmetry of the tumor cells.
- **fractal_dimension_mean**: Mean fractal dimension ("coastline approximation") of the tumor cells.

Standard Error Features:

- **radius_se**: Standard error of the radius of the tumor cells.
- **texture_se**: Standard error of the texture of the tumor cells.
- **perimeter_se**: Standard error of the perimeter of the tumor cells.
- **area_se**: Standard error of the area of the tumor cells.
- **smoothness_se**: Standard error of the smoothness of the tumor cells.
- **compactness_se**: Standard error of the compactness of the tumor cells.
- **concavity_se**: Standard error of the concavity of the tumor cells.
- **concave points_se**: Standard error of the number of concave portions of the tumor cell contours.
- **symmetry_se**: Standard error of the symmetry of the tumor cells.
- **fractal_dimension_se**: Standard error of the fractal dimension of the tumor cells.

Worst (Largest) Features:

- **radius_worst**: Largest (worst) radius of the tumor cells.
- **texture_worst**: Largest (worst) texture of the tumor cells.
- **perimeter_worst**: Largest (worst) perimeter of the tumor cells.
- **area_worst**: Largest (worst) area of the tumor cells.
- **smoothness_worst**: Largest (worst) smoothness of the tumor cells.
- **compactness_worst**: Largest (worst) compactness of the tumor cells.
- **concavity_worst**: Largest (worst) concavity of the tumor cells.
- **concave points_worst**: Largest (worst) number of concave portions of the tumor cell contours.
- **symmetry_worst**: Largest (worst) symmetry of the tumor cells.
- **fractal_dimension_worst**: Largest (worst) fractal dimension of the tumor cells.

Import Libraries

In [51]:
```
#  Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, classification_report, confusion_matrix
import warnings
warnings.filterwarnings("ignore")
```

Load the Cancer Wisconsin dataset

In [9]:
```
df = pd.read_csv('Cancer Wisconsin.csv')
```

In [11]:
```
df.head()
```

Out[11]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavit
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

Information of dataset

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

Out[14]:
```
(569, 33)
```

In [16]:
```
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
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

Check The Column Names

In [19]:
```
df.columns
```

Out[19]:
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

Data Cleaning

In [24]:
```
# Check for missing values
print("\nMissing Values:\n", df.isnull().sum())
```

```
Missing Values:
 id                          0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
Unnamed: 32              569
dtype: int64
```

In [26]:
```
# Drop Unwanted Columns
df.drop(['id', 'Unnamed: 32'], axis=1, inplace=True)
```

In [28]:
```
# Check the balance of the target classes
df['diagnosis'].value_counts()
```

Out[28]:
```
diagnosis
B    357
M    212
Name: count, dtype: int64
```

In [30]:
```
# Change The Diagnosis in Numeric (M=1, B=0)
df['diagnosis'] = df['diagnosis'].map({'M':1, 'B':0})
```

Check The Duplicate Values

In [33]:
```
# check the duplicate values
df.duplicated().sum()
```

Out[33]:
```
0
```

Summary Statistics

In [36]:
```
df.describe().T
```

Out[36]:

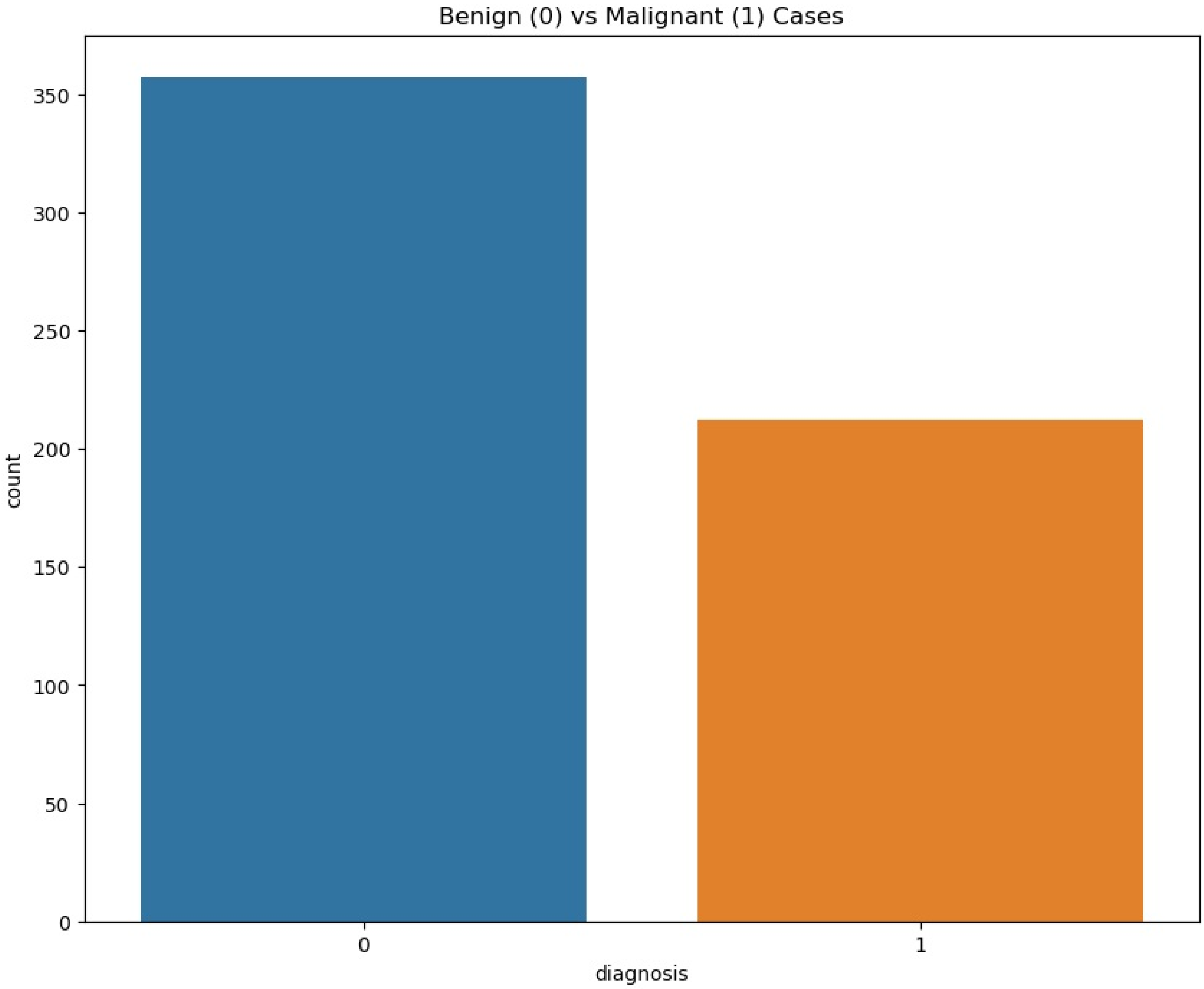
	count	mean	std	min	25%	50%	75%	max
diagnosis	569.0	0.372583	0.483918	0.000000	0.000000	0.000000	1.000000	1.00000
radius_mean	569.0	14.127292	3.524049	6.981000	11.700000	13.370000	15.780000	28.11000
texture_mean	569.0	19.289649	4.301036	9.710000	16.170000	18.840000	21.800000	39.28000
perimeter_mean	569.0	91.969033	24.298981	43.790000	75.170000	86.240000	104.100000	188.50000
area_mean	569.0	654.889104	351.914129	143.500000	420.300000	551.100000	782.700000	2501.00000
smoothness_mean	569.0	0.096360	0.014064	0.052630	0.086370	0.095870	0.105300	0.16340
compactness_mean	569.0	0.104341	0.052813	0.019380	0.064920	0.092630	0.130400	0.34540
concavity_mean	569.0	0.088799	0.079720	0.000000	0.029560	0.061540	0.130700	0.42680
concave points_mean	569.0	0.048919	0.038803	0.000000	0.020310	0.033500	0.074000	0.20120
symmetry_mean	569.0	0.181162	0.027414	0.106000	0.161900	0.179200	0.195700	0.30400
fractal_dimension_mean	569.0	0.062798	0.007060	0.049960	0.057700	0.061540	0.066120	0.09744
radius_se	569.0	0.405172	0.277313	0.111500	0.232400	0.324200	0.478900	2.87300
texture_se	569.0	1.216853	0.551648	0.360200	0.833900	1.108000	1.474000	4.88500
perimeter_se	569.0	2.866059	2.021855	0.757000	1.606000	2.287000	3.357000	21.98000
area_se	569.0	40.337079	45.491006	6.802000	17.850000	24.530000	45.190000	542.20000
smoothness_se	569.0	0.007041	0.003003	0.001713	0.005169	0.006380	0.008146	0.03113
compactness_se	569.0	0.025478	0.017908	0.002252	0.013080	0.020450	0.032450	0.13540
concavity_se	569.0	0.031894	0.030186	0.000000	0.015090	0.025890	0.042050	0.39600
concave points_se	569.0	0.011796	0.006170	0.000000	0.007638	0.010930	0.014710	0.05279
symmetry_se	569.0	0.020542	0.008266	0.007882	0.015160	0.018730	0.023480	0.07895
fractal_dimension_se	569.0	0.003795	0.002646	0.000895	0.002248	0.003187	0.004558	0.02984
radius_worst	569.0	16.269190	4.833242	7.930000	13.010000	14.970000	18.790000	36.04000
texture_worst	569.0	25.677223	6.146258	12.020000	21.080000	25.410000	29.720000	49.54000
perimeter_worst	569.0	107.261213	33.602542	50.410000	84.110000	97.660000	125.400000	251.20000
area_worst	569.0	880.583128	569.356993	185.200000	515.300000	686.500000	1084.000000	4254.00000
smoothness_worst	569.0	0.132369	0.022832	0.071170	0.116600	0.131300	0.146000	0.22260
compactness_worst	569.0	0.254265	0.157336	0.027290	0.147200	0.211900	0.339100	1.05800
concavity_worst	569.0	0.272188	0.208624	0.000000	0.114500	0.226700	0.382900	1.25200
concave points_worst	569.0	0.114606	0.065732	0.000000	0.064930	0.099930	0.161400	0.29100
symmetry_worst	569.0	0.290076	0.061867	0.156500	0.250400	0.282200	0.317900	0.66380
fractal_dimension_worst	569.0	0.083946	0.018061	0.055040	0.071460	0.080040	0.092080	0.20750

In [ ]:

Data Exploration

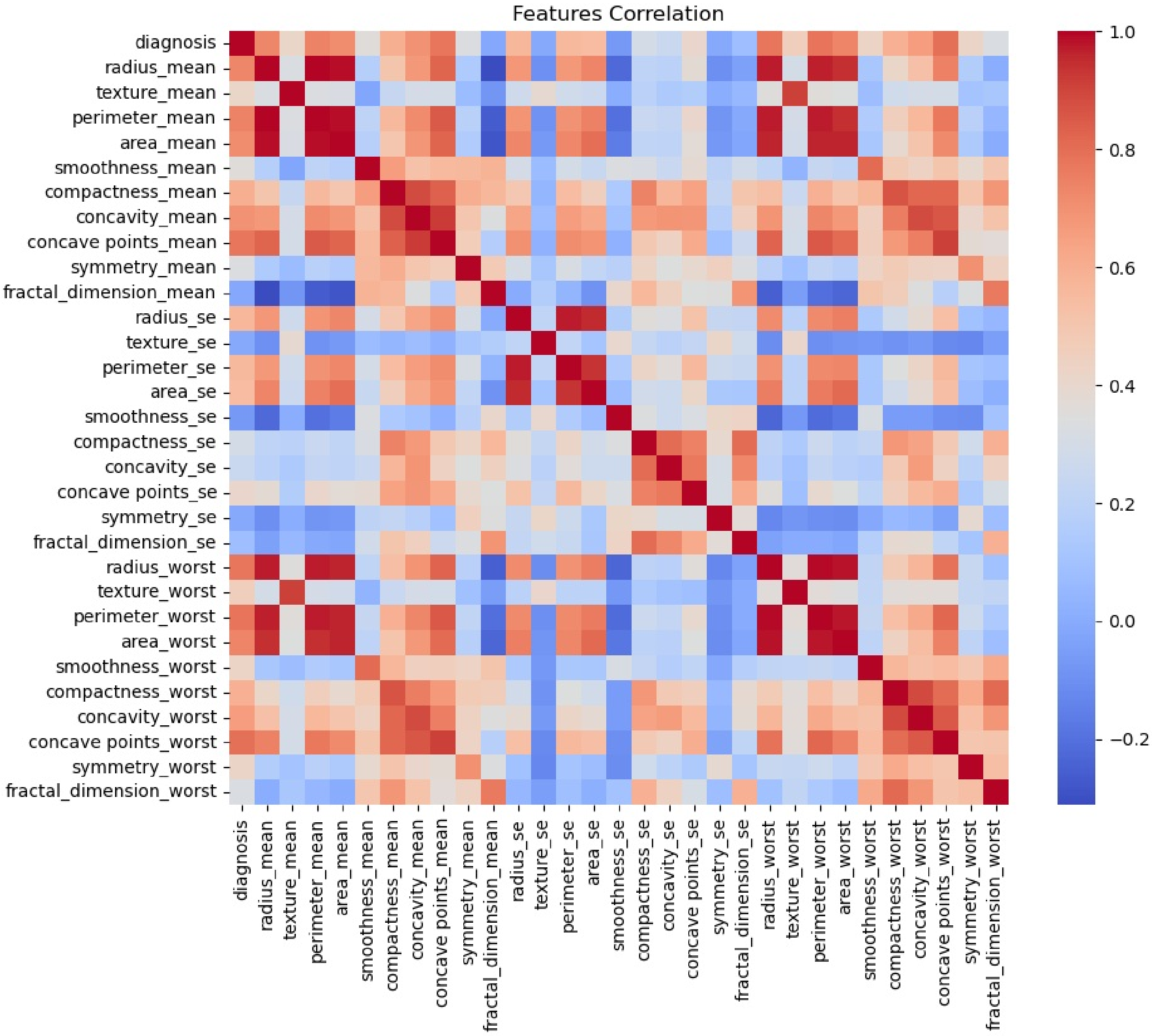
In [42]:

```
# 4.1 Diagnosis Count Plot
plt.figure(figsize=(10,8))
sns.countplot(x='diagnosis', data=df)
plt.title("Benign (0) vs Malignant (1) Cases")
plt.show()
```



Correlation Matrix

In [44]:
```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), annot=False, cmap='coolwarm')
plt.title("Features Correlation")
plt.show()
```



Data Preprocessing

In [46]:
```
# Features (X) and Target (y)
X = df.drop('diagnosis', axis=1)
y = df['diagnosis']

#  Feature Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

#  Train-Test Split (80% Train, 20% Test)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

Defining & Training All The Model For Choosing the Best Model

In [53]:
```
# Test Of Multiple Models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "Support Vector Machine": SVC(),
    "Naive Bayes": GaussianNB(),

}
```

In [55]:
```
results = []
for name, model in models.items():
    # Model Training
    model.fit(X_train, y_train)

    # For Predictions
```

```python
    y_pred = model.predict(X_test)

    # Check Performance
    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print("*"*60)
    results.append([name, acc, f1])

    # Classification Report
    print(f"\nModel: {name} \n")
    print("Accuracy:", round(acc, 4))
    print("F1 Score:", round(f1, 4))
    print("*"*60)
    print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
************************************************************

Model: Logistic Regression

Accuracy: 0.9737
F1 Score: 0.9647
************************************************************
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.99      0.98        71
           1       0.98      0.95      0.96        43

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114

************************************************************

Model: Decision Tree

Accuracy: 0.9474
F1 Score: 0.9302
************************************************************
Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.96      0.96        71
           1       0.93      0.93      0.93        43

    accuracy                           0.95       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114

************************************************************

Model: Random Forest

Accuracy: 0.9561
F1 Score: 0.9412
************************************************************
Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.97      0.97        71
           1       0.95      0.93      0.94        43

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114

************************************************************

Model: Support Vector Machine

Accuracy: 0.9737
F1 Score: 0.9647
************************************************************
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.99      0.98        71
           1       0.98      0.95      0.96        43

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114

************************************************************
```

```
Model: Naive Bayes

Accuracy: 0.9649
F1 Score: 0.9524
************************************************************
Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.99      0.97        71
           1       0.98      0.93      0.95        43

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

Results Comparison

In [58]:

```python
# Summary table
# Results Comparison
results_df = pd.DataFrame(results, columns=['Model', 'Accuracy', 'F1 Score'])
print("\nModels Comparison:")
results_df.sort_values('F1 Score', ascending=False)
```

```
Models Comparison:
```

Out[58]:

	Model	Accuracy	F1 Score
0	Logistic Regression	0.973684	0.964706
3	Support Vector Machine	0.973684	0.964706
4	Naive Bayes	0.964912	0.952381
2	Random Forest	0.956140	0.941176
1	Decision Tree	0.947368	0.930233

Model Performance

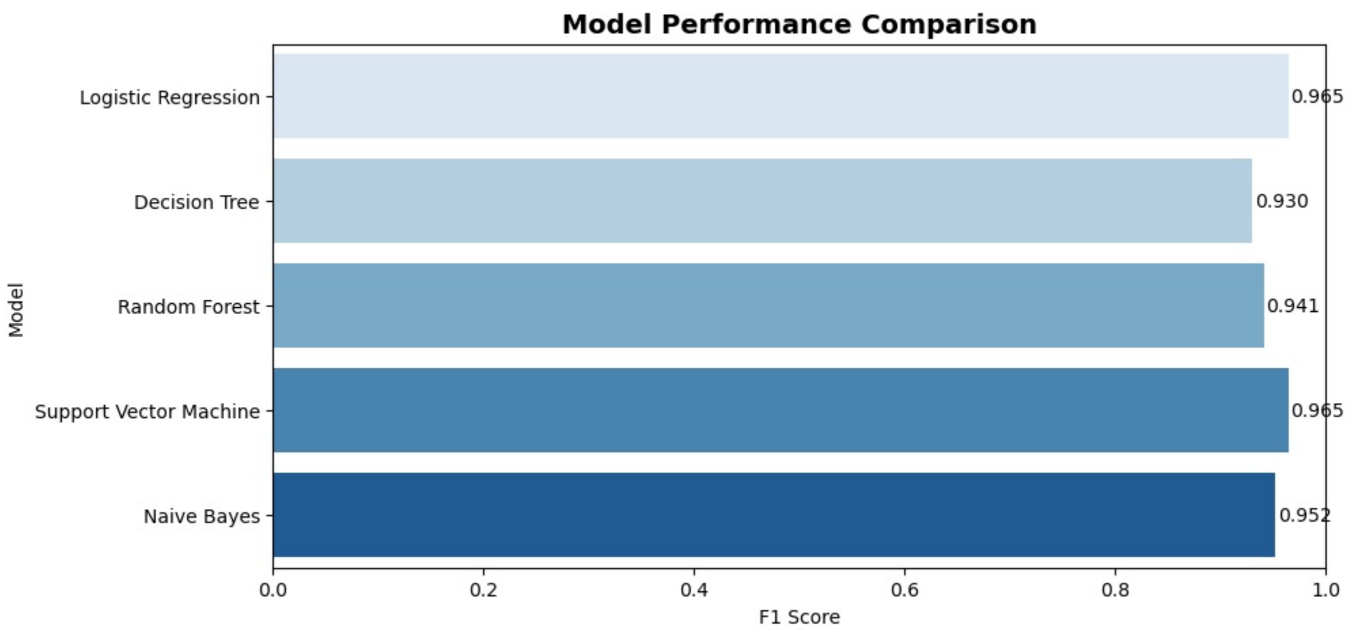
In [61]:

```python
import seaborn as sns
# Plot Model Performance
plt.figure(figsize=(10, 5))
ax = sns.barplot(x="F1 Score", y="Model", data=results_df, palette="Blues")

# Add annotations (F1 Score on bars)
for container in ax.containers:
    ax.bar_label(container, fmt="%.3f", fontsize=10, color="black", padding=1)

# Set title and limits
plt.title("Model Performance Comparison", fontsize=14, fontweight="bold")
plt.xlim(0, 1)  #F1 Score range

# Show plot
plt.show()
```



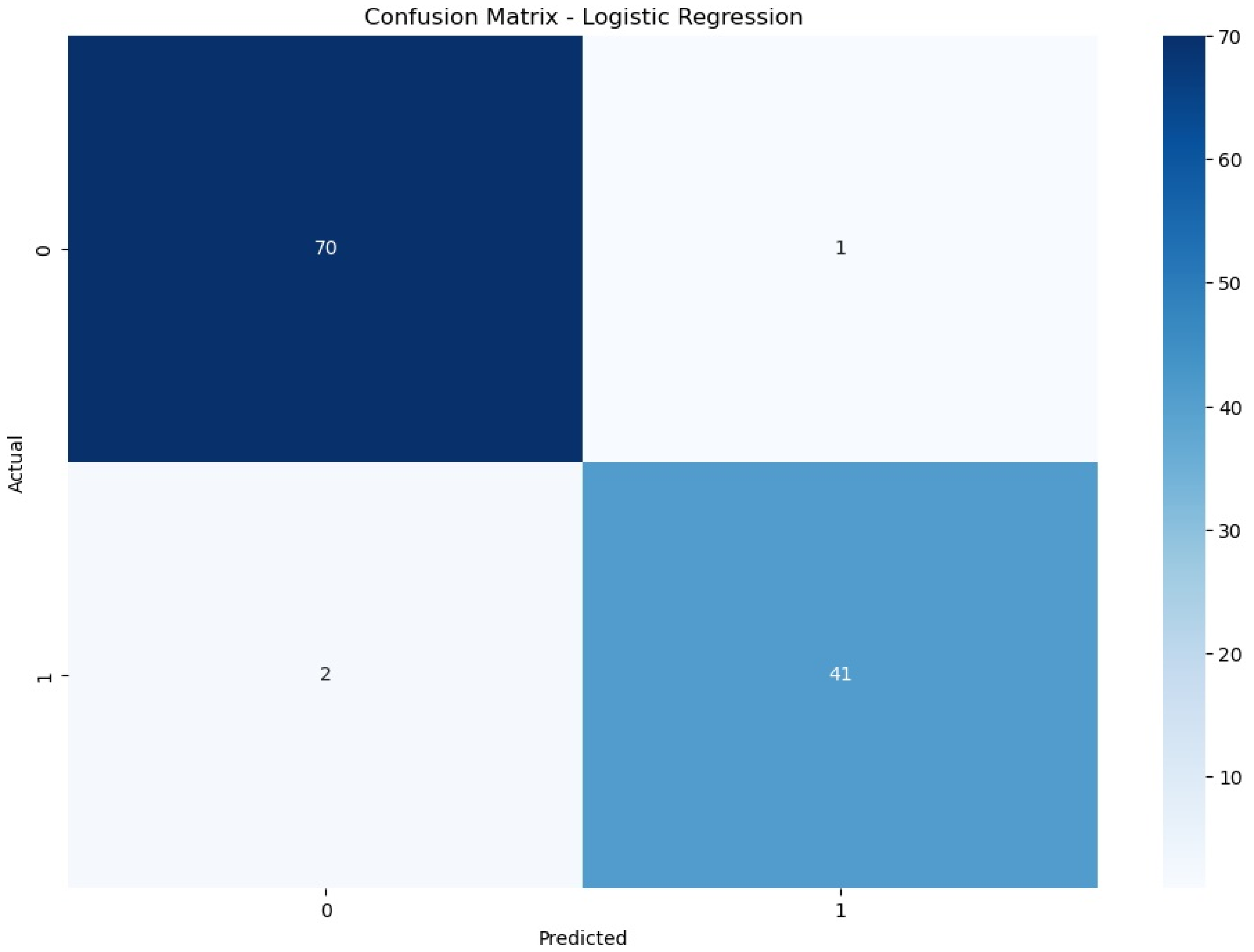
Choose The Best Model

- Logistic Regression Model is the best model for batter prediction

Confusion Matrix

In [65]:
```
#  Best Model ka Confusion Matrix
plt.figure(figsize=(12, 8))
best_model = LogisticRegression(max_iter=1000)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)


cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Logistic Regression")
plt.show()
```



In [ ]:

New Patient Per Prediction

In [69]:
```
# New Patient Per Prediction
# Sample New Data
new_patient = [[17.99, 10.38, 122.8, 1001.0, 0.1184, 0.2776, 0.3001, 0.1471, 0.2419, 0.07871,
                1.095, 0.9053, 8.589, 153.4, 0.006399, 0.04904, 0.05373, 0.01587, 0.03003, 0.006193,
                25.38, 17.33, 184.6, 2019.0, 0.1622, 0.6656, 0.7119, 0.2654, 0.4601, 0.1189]]

#  Feature Scaling for New Data
new_patient_scaled = scaler.transform(new_patient)

# Prediction
prediction = best_model.predict(new_patient_scaled)
```

```python
prediction_proba = best_model.predict_proba(new_patient_scaled)

#  Result Display
print("\nNew Patient Prediction:")
print("Predicted Class:", "Malignant (Cancer)" if prediction[0] == 1 else "Benign (No Cancer)")
print("Probability [Benign, Malignant]:", prediction_proba[0])
```

```
New Patient Prediction:
Predicted Class: Malignant (Cancer)
Probability [Benign, Malignant]: [5.49004375e-09 9.99999995e-01]
```

In [ ]:

In [ ]:

Run The Model in Streamlit Web App

In [74]:
```python
%%writefile app.py
import streamlit as st
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
import pickle

# Load saved model and scaler
with open('cancer_model.pkl', 'rb') as model_file:
    model = pickle.load(model_file)
with open('scaler.pkl', 'rb') as scaler_file:
    scaler = pickle.load(scaler_file)

# App title and description
st.title(" Breast Cancer Prediction App")
st.write("""
This app predicts whether a breast tumor is **Malignant (Cancerous)** or **Benign (Non-Cancerous)**
using machine learning. Enter the patient's details below:
""")

# Input form
st.header("Patient Details")
with st.form("prediction_form"):
    # Create input fields for all 30 features
    col1, col2 = st.columns(2)

    with col1:
        radius_mean = st.number_input("Radius Mean", min_value=0.0, value=17.99)
        texture_mean = st.number_input("Texture Mean", min_value=0.0, value=10.38)
        perimeter_mean = st.number_input("Perimeter Mean", min_value=0.0, value=122.8)
        area_mean = st.number_input("Area Mean", min_value=0.0, value=1001.0)
        smoothness_mean = st.number_input("Smoothness Mean", min_value=0.0, value=0.1184)

    with col2:
        compactness_mean = st.number_input("Compactness Mean", min_value=0.0, value=0.2776)
        concavity_mean = st.number_input("Concavity Mean", min_value=0.0, value=0.3001)
        concave_points_mean = st.number_input("Concave Points Mean", min_value=0.0, value=0.1471)
        symmetry_mean = st.number_input("Symmetry Mean", min_value=0.0, value=0.2419)
        fractal_dimension_mean = st.number_input("Fractal Dimension Mean", min_value=0.0, value=0.07871)

    submit_button = st.form_submit_button("Predict Diagnosis")

# Prediction logic
if submit_button:
    # Create feature array
    features = np.array([[
        radius_mean, texture_mean, perimeter_mean, area_mean, smoothness_mean,
        compactness_mean, concavity_mean, concave_points_mean, symmetry_mean,
        fractal_dimension_mean,
        # Add remaining features with default values
        1.095, 0.9053, 8.589, 153.4, 0.006399, 0.04904, 0.05373, 0.01587, 0.03003, 0.006193,
        25.38, 17.33, 184.6, 2019.0, 0.1622, 0.6656, 0.7119, 0.2654, 0.4601, 0.1189
    ]])

    # Scale features
    features_scaled = scaler.transform(features)

    # Make prediction
    prediction = model.predict(features_scaled)
    probability = model.predict_proba(features_scaled)

    # Display results
    st.header("Prediction Results")
```

```
    if prediction[0] == 1:
        st.error(f" **Prediction:** Malignant (Cancerous) - {probability[0][1]*100:.2f}% probability")
    else:
        st.success(f"✅ **Prediction:** Benign (Non-Cancerous) - {probability[0][0]*100:.2f}% probability")

    # Show probability breakdown
    st.write(f"**Probability Breakdown:**")
    st.write(f"- Benign: {probability[0][0]*100:.2f}%")
    st.write(f"- Malignant: {probability[0][1]*100:.2f}%")

# Run instructions
st.sidebar.header("How to Use")
st.sidebar.write("""
1. Enter patient's tumor characteristics
2. Click 'Predict Diagnosis'
3. View results
""")

# Note: For simplicity, I've included only 10 input fields.
# You should add all 30 features for complete functionality.
```

Overwriting app.py

In [76]:

```
import subprocess
import sys

# Install streamlit if not installed
subprocess.check_call([sys.executable, "-m", "pip", "install", "streamlit"])

# Run the streamlit app
subprocess.Popen([sys.executable, "-m", "streamlit", "run", "app.py"])
```

Out[76]:

```
<Popen: returncode: None args: ['C:\\ProgramData\\anaconda3\\python.exe', '-...>
```

In [ ]: