In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [22]:
```
df = pd.read_csv("D:/D.Desktop/New Project/Breast Cancer/data.csv")
df.head()
```

Out[22]:

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

5 rows × 33 columns

In [23]:
```
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
```

In [24]:
```
df.describe()
```

Out[24]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactn
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	5
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	

8 rows × 32 columns

In [25]:
```
df.isnull().sum()
```

Out[25]:
```
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]:
```
df = df.drop(['id','Unnamed: 32'], axis = 1)
```

In [27]:
```
df.head()
```

Out[27]:

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

5 rows × 31 columns

In [28]:
```
df.duplicated().sum()
```

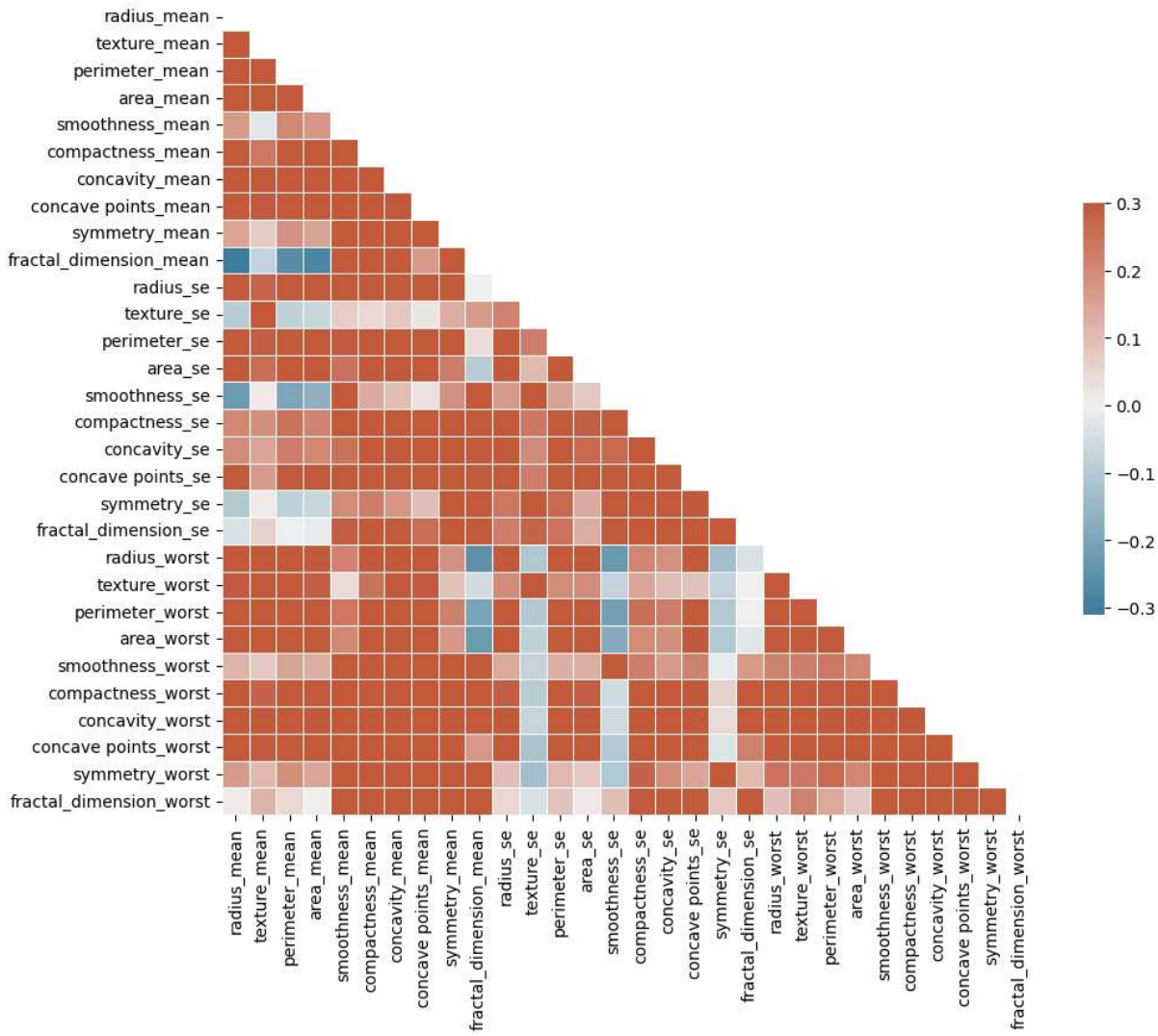
Out[28]:
```
0
```

In [29]:
```
import ipywidgets as widgets
```

In [30]:
```
# Define a function to plot the disctribution of features
def plot_feature(feature):
    plt.figure(figsize=(10,6))
    df[feature].hist(bins=30)
    plt.title(f'Distribution of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.show()

# Create a dropdown widget with the dataframe's column names
dropdown = widgets.Dropdown(options=df.columns, description='Feature:')

# Use the interact function to create the widget and the plot
widgets.interact(plot_feature, feature=dropdown);
```

```
interactive(children=(Dropdown(description='Feature:', options=('diagnosis', 'radius_me
an', 'texture_mean', 'p…
```

In [33]:
```
# Compute the correlation matrix
corr = df.corr(numeric_only=True)

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})

plt.show()
```



In [36]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
```

In [37]:
```
# Divide the dataframe into features (X) and target (y)
X = df.drop('diagnosis', axis=1)
y = df['diagnosis']
```

In [38]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [39]:
```
# Function to train and evaluate a model
def train_evaluate_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)
    return accuracy, report
```

In [40]:
```
# Define the models you want to train
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Random Forest': RandomForestClassifier(),
    'Support Vector Machine': SVC()
}
```

In [41]:
```
# Train and evaluate each model
results = {}
for model_name, model in models.items():
    accuracy, report = train_evaluate_model(model, X_train, X_test, y_train, y_test)
    results[model_name] = {'Accuracy': accuracy, 'Report': report}

# Print the results
for model_name, metrics in results.items():
    print(f"Model: {model_name}")
    print(f"Accuracy: {metrics['Accuracy']}")
    print(f"Classification Report:\n{metrics['Report']}\n")
```

```
D:\anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarnin
g: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.or
g/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (http
s://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

```
Model: Logistic Regression
Accuracy: 0.956140350877193
Classification Report:
              precision    recall  f1-score   support

           B       0.95      0.99      0.97        71
           M       0.97      0.91      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: Random Forest
Accuracy: 0.956140350877193
Classification Report:
              precision    recall  f1-score   support

           B       0.96      0.97      0.97        71
           M       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: 0.9473684210526315
Classification Report:
              precision    recall  f1-score   support

           B       0.92      1.00      0.96        71
           M       1.00      0.86      0.93        43

    accuracy                           0.95       114
   macro avg       0.96      0.93      0.94       114
weighted avg       0.95      0.95      0.95       114
```

In [ ]: