In [1]:
```python
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PowerTransformer
```

In [2]:
```python
# Load the Breast Cancer dataset
cancer = datasets.load_breast_cancer()
X, y = cancer.data, cancer.target
cancer
```

Out[2]:
```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
         8.758e-02],
        ...,
        [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
         7.039e-02]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
        1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
        1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]),
 'frame': None,
 'target_names': array(['malignant', 'benign'], dtype='<U9'),
 'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic) dataset\n--------------------------------------------\n\n**Data Set Characteristics:**\n\n
:Number of Instances: 569\n\n    :Number of Attributes: 30 numeric, predictive attributes and the class\n\n    :Attribute Information:\n        - radius (mean of distan
ces from center to points on the perimeter)\n        - texture (standard deviation of gray-scale values)\n        - perimeter\n        - area\n        - smoothness (loc
al variation in radius lengths)\n        - compactness (perimeter^2 / area - 1.0)\n        - concavity (severity of concave portions of the contour)\n        - concave
points (number of concave portions of the contour)\n        - symmetry\n        - fractal dimension ("coastline approximation" - 1)\n\n        The mean, standard error,
and "worst" or largest (mean of the three\n        worst/largest values) of these features were computed for each image,\n        resulting in 30 features.  For instanc
e, field 0 is Mean Radius, field\n        10 is Radius SE, field 20 is Worst Radius.\n\n        - class:\n                - WDBC-Malignant\n                - WDBC-Benig
n\n\n    :Summary Statistics:\n\n    ===================================== ====== ======\n                                           Min    Max\n    ===================
================= ====== ======\n    radius (mean):                        6.981  28.11\n    texture (mean):                       9.71   39.28\n    perimeter (mean):
43.79  188.5\n    area (mean):                          143.5  2501.0\n    smoothness (mean):                    0.053  0.163\n    compactness (mean):
0.019  0.345\n    concavity (mean):                     0.0    0.427\n    concave points (mean):                0.0    0.201\n    symmetry (mean):
0.106  0.304\n    fractal dimension (mean):             0.05   0.097\n    radius (standard error):              0.112  2.873\n    texture (standard error):
0.36   4.885\n    perimeter (standard error):           0.757  21.98\n    area (standard error):                6.802  542.2\n    smoothness (standard error):
0.002  0.031\n    compactness (standard error):         0.002  0.135\n    concavity (standard error):           0.0    0.396\n    concave points (standard error):
0.0    0.053\n    symmetry (standard error):            0.008  0.079\n    fractal dimension (standard error):   0.001  0.03\n    radius (worst):
7.93   36.04\n    texture (worst):                      12.02  49.54\n    perimeter (worst):                    50.41  251.2\n    area (worst):
185.2  4254.0\n    smoothness (worst):                   0.071  0.223\n    compactness (worst):                  0.027  1.058\n    concavity (worst):
0.0    1.252\n    concave points (worst):               0.0    0.291\n    symmetry (worst):                     0.156  0.664\n    fractal dimension (worst):
0.055  0.208\n    ===================================== ====== ======\n\n    :Missing Attribute Values: None\n\n    :Class Distribution: 212 - Malignant, 357 - Benign\n
\n    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian\n\n    :Donor: Nick Street\n\n    :Date: November, 1995\n\nThis is a copy of UCI ML Breast
Cancer Wisconsin (Diagnostic) datasets.\nhttps://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image of a fine needle\naspirate (FNA) of a breast mass.  They
describe\ncharacteristics of the cell nuclei present in the image.\n\nSeparating plane described above was obtained using\nMultisurface Method-Tree (MSM-T) [K. P. Benne
tt, "Decision Tree\nConstruction Via Linear Programming." Proceedings of the 4th\nMidwest Artificial Intelligence and Cognitive Science Society,\npp. 97-101, 1992], a c
lassification method which uses linear\nprogramming to construct a decision tree.  Relevant features\nwere selected using an exhaustive search in the space of 1-4\nfeat
ures and 1-3 separating planes.\n\nThe actual linear program used to obtain the separating plane\nin the 3-dimensional space is that described in:\n[K. P. Bennett and
O. L. Mangasarian: "Robust Linear\nProgramming Discrimination of Two Linearly Inseparable Sets",\nOptimization Methods and Software 1, 1992, 23-34].\n\nThis database is
also available through the UW CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-prog/cpo-dataset/machine-learn/WDBC/\n\n.. topic:: References\n\n   - W.N. Street, W.H. Wol
berg and O.L. Mangasarian. Nuclear feature extraction \n     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on \n     Electronic Imaging: Science an
d Technology, volume 1905, pages 861-870,\n     San Jose, CA, 1993.\n   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and \n     prognosis v
ia linear programming. Operations Research, 43(4), pages 570-577, \n     July-August 1995.\n   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techn
iques\n     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994) \n     163-171.',
 'feature_names': array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
        'mean smoothness', 'mean compactness', 'mean concavity',
        'mean concave points', 'mean symmetry', 'mean fractal dimension',
        'radius error', 'texture error', 'perimeter error', 'area error',
        'smoothness error', 'compactness error', 'concavity error',
        'concave points error', 'symmetry error',
        'fractal dimension error', 'worst radius', 'worst texture',
        'worst perimeter', 'worst area', 'worst smoothness',
        'worst compactness', 'worst concavity', 'worst concave points',
        'worst symmetry', 'worst fractal dimension'], dtype='<U23'),
 'filename': 'breast_cancer.csv',
 'data_module': 'sklearn.datasets.data'}
```

In [3]:
```python
# Introduce missing values for demonstration
X[10, 2] = np.nan
X[30, 1] = np.nan
```

In [4]:
```python
# Step 1: Handling Missing Values
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)
```

In [5]:
```
X_imputed
```

Out[5]:
```
array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
        8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
        8.758e-02],
       ...,
       [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
        7.820e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
        1.240e-01],
       [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
        7.039e-02]])
```

In [6]:
```
# Step 3: Normalization or Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)
```

In [7]:
```
# Step 4: Feature Engineering
# Introduce an artificial time column
time_column = np.arange(X_scaled.shape[0]).reshape(-1, 1)
X_with_time = np.hstack((X_scaled, time_column))
```

In [8]:
```
# Create time-based features
X_with_time = pd.DataFrame(X_with_time, columns=[f'feature_{i}' for i in range(X_scaled.shape[1])] + ['time'])
X_with_time['month'] = X_with_time['time'] % 12 + 1
X_with_time['day_of_week'] = X_with_time['time'] % 7 + 1
```

In [9]:
```
# Rolling statistics
rolling_window = 5
for i in range(X_scaled.shape[1]):
    X_with_time[f'rolling_mean_{i}'] = X_with_time[f'feature_{i}'].rolling(window=rolling_window).mean()
    X_with_time[f'rolling_std_{i}'] = X_with_time[f'feature_{i}'].rolling(window=rolling_window).std()
```

In [10]:
```
X_with_time
```

Out[10]:

	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	...	rolling_mean_25	rolling_std_25	rolling_mean_26	rolling_std_26	rolling_mean_27	rolling_std_2
0	1.097064	-2.074303	1.270930	0.984375	1.568466	3.283515	2.652874	2.532475	2.217515	2.255747	...	NaN	NaN	NaN	NaN	NaN	Na
1	1.829821	-0.351816	1.687023	1.908708	-0.826962	-0.487072	-0.023846	0.548144	0.001392	-0.868652	...	NaN	NaN	NaN	NaN	NaN	Na
2	1.579888	0.459314	1.567551	1.558884	0.942210	1.052926	1.363478	2.037231	0.939685	-0.398008	...	NaN	NaN	NaN	NaN	NaN	Na
3	-0.768909	0.256531	-0.592011	-0.764464	3.283553	3.402909	1.915897	1.451707	2.867383	4.910919	...	NaN	NaN	NaN	NaN	NaN	Na
4	1.750297	-1.151293	1.777657	1.826229	0.280372	0.539340	1.371011	1.428493	-0.009560	-0.562450	...	1.369831	1.876135	1.084104	0.956643	1.648641	0.69848
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	2.110995	0.725029	2.061918	2.343856	1.041842	0.219060	1.947285	2.320965	-0.312589	-0.931027	...	0.549135	1.801907	0.971266	2.229958	0.740527	1.65013
565	1.704854	2.090898	1.616988	1.723842	0.102458	-0.017833	0.693043	1.263669	-0.217664	-1.058611	...	0.505623	1.826295	1.152516	2.096048	0.917155	1.57671
566	0.702284	2.051274	0.673570	0.577953	-0.840484	-0.038680	0.046588	0.105777	-0.809117	-0.895587	...	0.829367	1.556893	1.479036	1.708726	1.348982	0.73780
567	1.838341	2.342628	1.983644	1.735218	1.525767	3.272144	3.296944	2.658866	2.137194	1.043695	...	0.926569	1.760799	1.257102	1.263652	1.438514	0.83358
568	-1.808401	1.226158	-1.813922	-1.347789	-3.112085	-1.150752	-1.114873	-1.261820	-0.820070	-0.561032	...	0.475979	1.995422	0.623925	1.626943	0.664394	1.53673

569 rows × 93 columns

In [11]:
```
# Additional transformations
X_with_time['log_feature_0'] = np.log1p(X_with_time['feature_0'])
X_with_time['sqrt_feature_1'] = np.sqrt(X_with_time['feature_1'])
```

```
C:\Users\Chiranjeevi\anaconda3\lib\site-packages\pandas\core\arraylike.py:358: RuntimeWarning: invalid value encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
C:\Users\Chiranjeevi\anaconda3\lib\site-packages\pandas\core\arraylike.py:358: RuntimeWarning: invalid value encountered in sqrt
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

In [12]:
```
# Drop rows with missing values introduced during rolling statistics computation
X_with_time = X_with_time.dropna()
```

In [13]:
```
X_with_time
```

Out[13]:

	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	...	rolling_mean_26	rolling_std_26	rolling_mean_27	rolling_std_27	rolling_mean_28	rolling_s
6	1.170908	0.163298	1.139099e+00	1.095295	-0.123136	0.088295	0.300072	0.646935	-0.064325	-0.762332	...	1.046107	0.601904	1.392530	0.641279	1.669298	2.6
7	-0.118517	0.361419	-7.210105e-02	-0.218965	1.604049	1.140102	0.061026	0.281950	1.403355	1.660353	...	0.870902	0.774269	1.126369	0.625442	1.534375	2.6
8	-0.320167	0.592171	-1.833337e-01	-0.384207	2.201839	1.684010	1.219096	1.150692	1.965600	1.572462	...	0.728991	0.550728	0.969535	0.320471	0.803138	1.2
9	-0.473535	1.109617	-3.287601e-01	-0.509063	1.582699	2.563358	1.738872	0.941760	0.797298	2.783096	...	1.405442	1.547701	1.147686	0.392909	1.450897	1.0
10	0.537556	0.923150	5.854486e-16	0.406453	-1.017686	-0.713542	-0.700684	-0.404686	-1.035476	-0.826124	...	1.031619	1.796382	0.921267	0.739984	1.115370	1.1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
563	1.929226	1.354354	2.103116e+00	1.968434	0.963560	2.260135	2.870075	2.540213	1.231760	0.849484	...	0.925498	2.240153	0.359649	1.613120	-0.504029	1.5
564	2.110995	0.725029	2.061918e+00	2.343856	1.041842	0.219060	1.947285	2.320965	-0.312589	-0.931027	...	0.971266	2.229958	0.740527	1.650139	-0.520854	1.5
565	1.704854	2.090898	1.616988e+00	1.723842	0.102458	-0.017833	0.693043	1.263669	-0.217664	-1.058611	...	1.152516	2.096048	0.917155	1.576712	-0.416669	1.5
566	0.702284	2.051274	6.735698e-01	0.577953	-0.840484	-0.038680	0.046588	0.105777	-0.809117	-0.895587	...	1.479036	1.708726	1.348982	0.737801	-0.205710	1.3
567	1.838341	2.342628	1.983644e+00	1.735218	1.525767	3.272144	3.296944	2.658866	2.137194	1.043695	...	1.257102	1.263652	1.438514	0.833580	-0.206357	1.3

239 rows × 95 columns

In [14]:
```
# Drop rows with missing values introduced during rolling statistics computation
X_with_time = X_with_time.dropna()

# Update the target variable 'y' accordingly
y = y[:X_with_time.shape[0]]

# Split the Data
X_train, X_test, y_train, y_test = train_test_split(X_with_time.drop(['time'], axis=1), y, test_size=0.2, random_state=42)
```

In [15]:
```
X_train
```

Out[15]:

	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	...	rolling_mean_26	rolling_std_26	rolling_mean_27	rolling_std_27	rolling_mean_28	rolling_std_2
302	1.693494	1.067662	1.761178	1.684024	0.828346	1.505866	1.751427	2.039810	1.596855	1.685870	...	0.127458	1.160639	0.000210	1.153491	-0.224476	0.64381
119	1.085703	0.170290	0.916634	0.930337	-0.878202	-0.703498	-0.199239	0.181612	1.158741	-1.778754	...	0.359486	1.215894	0.190329	1.140789	0.429431	1.97376
506	-0.541698	0.177283	-0.514148	-0.573909	0.942210	0.205794	-0.088504	-0.703122	1.140487	0.870748	...	0.059545	0.405083	0.086362	0.919967	0.049899	0.27824
413	0.245021	0.659766	0.229875	0.110382	-0.797785	-0.034889	-0.253727	-0.262045	0.483317	-0.519922	...	-0.419536	0.374416	-0.487194	0.580680	0.149231	0.63869
503	2.545536	0.128335	2.478011	2.921209	-0.209246	0.438898	0.989340	1.325317	-1.119447	-1.128074	...	0.238876	0.671016	0.617222	0.966967	-0.042961	0.93106
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
232	-0.825712	3.386842	-0.871740	-0.762473	-1.320851	-1.300090	-1.052512	-1.095861	0.121873	-0.640418	...	0.280403	1.235289	0.145624	1.558811	0.193558	0.39333
26	0.128576	0.524577	0.224931	-0.028694	0.643316	1.562720	0.674211	1.003666	1.607807	0.913276	...	0.953515	0.606083	1.832579	0.448057	1.594556	1.14161
201	0.969258	0.009463	0.953711	0.843876	-0.475405	0.292971	0.185822	0.669634	-1.115796	-1.118151	...	0.253556	0.653073	0.533780	0.708151	0.455315	1.50415
446	1.028901	2.039619	1.044345	0.929199	0.256887	0.512808	1.016961	0.877275	-0.360051	-0.515669	...	-0.053840	1.273178	-0.143103	1.156278	-0.495940	0.70315
223	0.460872	0.226230	0.438745	0.302644	0.436936	0.304342	0.325182	0.404988	0.450458	0.032944	...	-0.061910	0.691968	-0.107869	0.695130	0.227208	0.97456

191 rows × 94 columns

In [16]:
```
# Step 6: Pipeline with PCA and RandomForestClassifier
pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=10)),
    ('classifier', RandomForestClassifier(random_state=42))
])
```

In [17]:
```
# Fit the model
pipeline.fit(X_train, y_train)
```

Out[17]:
```
Pipeline
SimpleImputer
StandardScaler
PCA
RandomForestClassifier
```

In [18]:
```
# Step 8: Monitor Performance Metrics
y_pred = pipeline.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.42
```

In [ ]: