Assignment4_ANN

October 27, 2024

1 Semiconductor manufacturing process dataset

1.1 Project Description

Source: https://www.kaggle.com/saurabhbagchi/fmst-semiconductor-manufacturing-project

A complex modern semiconductor manufacturing process is normally under constant surveillance via the monitoring of signals/variables collected from sensors and or process measurement points. However, not all of these signals are equally valuable in a specific monitoring system. The measured signals contain a combination of useful information, irrelevant information as well as noise. Engineers typically have a much larger number of signals than are actually required. If we consider each type of signal as a feature, then feature selection may be applied to identify the most relevant signals. The Process Engineers may then use these signals to determine key factors contributing to yield excursions downstream in the process. This will enable an increase in process throughput, decreased time to learning, and reduce per-unit production costs. These signals can be used as features to predict the yield type. And by analyzing and trying out different combinations of features, essential signals that are impacting the yield type can be identified.

Dataset: SemiconductorManufacturingProcessDataset.csv (on Canvas)

Later, we will learn how to apply PCA (Principal Component Analyses) for feature selection; then we will apply ANN to predict the Pass/Fail. in this exercise our objective is to repeat the same steps we did above for Supplier Data: Cleaning & Scaling Data, Encode Categorical Data, Split the Data to Training & Test Sets.

1.2 Importing the Libraries

[188]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

1.3 Importing the Dataset

[189]:
```
dataset = pd.read_csv('SemiconductorManufacturingProcessDataset.csv')
```

1.4 Showing the Dataset in a Table

[190]:
```python
pd.DataFrame(dataset)
#dataset
```

[190]:
```
                  Time  Sensor 1  Sensor 2  Sensor 3   Sensor 4  Sensor 5  \
0      7/19/2008 11:55   3030.93   2564.00  2187.7333  1411.1265    1.3602
1      7/19/2008 12:32   3095.78   2465.14  2230.4222  1463.6606    0.8294
2      7/19/2008 13:17   2932.61   2559.94  2186.4111  1698.0172    1.5102
3      7/19/2008 14:43   2988.72   2479.90  2199.0333   909.7926    1.3204
4      7/19/2008 15:22   3032.24   2502.87  2233.3667  1326.5200    1.5334
…                  …        …        …         …          …         …
1562  10/16/2008 15:13   2899.41   2464.36  2179.7333  3085.3781    1.4843
1563  10/16/2008 20:49   3052.31   2522.55  2198.5667  1124.6595    0.8763
1564   10/17/2008 5:26   2978.81   2379.78  2206.3000  1110.4967    0.8236
1565   10/17/2008 6:01   2894.92   2532.01  2177.0333  1183.7287    1.5726
1566   10/17/2008 6:07   2944.92   2450.76  2195.4444  2914.1792    1.5978

      Sensor 6  Sensor 7  Sensor 8  Sensor 9  …  Sensor 429  Sensor 430  \
0      97.6133    0.1242    1.5005    0.0162  …     14.9509      0.5005
1     102.3433    0.1247    1.4966   -0.0005  …     10.9003      0.5019
2      95.4878    0.1241    1.4436    0.0041  …      9.2721      0.4958
3     104.2367    0.1217    1.4882   -0.0124  …      8.5831      0.4990
4     100.3967    0.1235    1.5031   -0.0031  …     10.9698      0.4800
…          …        …        …        …  …           …
1562   82.2467    0.1248    1.3424   -0.0045  …     11.7256      0.4988
1563   98.4689    0.1205    1.4333   -0.0061  …     17.8379      0.4975
1564   99.4122    0.1208       NaN       NaN  …     17.7267      0.4987
1565   98.7978    0.1213    1.4622   -0.0072  …     19.2104      0.5004
1566   85.1011    0.1235       NaN       NaN  …     22.9183      0.4987

      Sensor 431  Sensor 432  Sensor 433  Sensor 434  Sensor 435  Sensor 436  \
0         0.0118      0.0035      2.3630         NaN         NaN         NaN
1         0.0223      0.0055      4.4447      0.0096      0.0201      0.0060
2         0.0157      0.0039      3.1745      0.0584      0.0484      0.0148
3         0.0103      0.0025      2.0544      0.0202      0.0149      0.0044
4         0.4766      0.1045     99.3032      0.0202      0.0149      0.0044
…            …           …           …           …           …           …
1562      0.0143      0.0039      2.8669      0.0068      0.0138      0.0047
1563      0.0131      0.0036      2.6238      0.0068      0.0138      0.0047
1564      0.0153      0.0041      3.0590      0.0197      0.0086      0.0025
1565      0.0178      0.0038      3.5662      0.0262      0.0245      0.0075
1566      0.0181      0.0040      3.6275      0.0117      0.0162      0.0045

      Sensor 437  Pass/Fail
0            NaN       Pass
1       208.2045       Pass
```

```
2        82.8602       Fail
3        73.8432       Pass
4        73.8432       Pass
…           …        …
1562    203.1720       Pass
1563    203.1720       Pass
1564     43.5231       Pass
1565     93.4941       Pass
1566    137.7844       Pass

[1567 rows x 439 columns]
```

1.5 A Quick Review of the Data

[191]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1567 entries, 0 to 1566
Columns: 439 entries, Time to Pass/Fail
dtypes: float64(437), object(2)
memory usage: 5.2+ MB
```

1.6 Seperate The Input and Output

Here, we put the independent variables in X and the dependent variable in y.

[192]:
```
X = dataset.iloc[:, 1:438].values
y = dataset.iloc[:, -1].values
```

1.7 Showing the Input Data in a Table format

[193]:
```
pd.DataFrame(X)
```

[193]:
```
            0        1          2          3       4         5       6   \
0     3030.93  2564.00  2187.7333  1411.1265  1.3602   97.6133  0.1242
1     3095.78  2465.14  2230.4222  1463.6606  0.8294  102.3433  0.1247
2     2932.61  2559.94  2186.4111  1698.0172  1.5102   95.4878  0.1241
3     2988.72  2479.90  2199.0333   909.7926  1.3204  104.2367  0.1217
4     3032.24  2502.87  2233.3667  1326.5200  1.5334  100.3967  0.1235
…         …        …          …        …         …        …      …
1562  2899.41  2464.36  2179.7333  3085.3781  1.4843   82.2467  0.1248
1563  3052.31  2522.55  2198.5667  1124.6595  0.8763   98.4689  0.1205
1564  2978.81  2379.78  2206.3000  1110.4967  0.8236   99.4122  0.1208
1565  2894.92  2532.01  2177.0333  1183.7287  1.5726   98.7978  0.1213
1566  2944.92  2450.76  2195.4444  2914.1792  1.5978   85.1011  0.1235

          7       8       9   …     427      428     429     430     431  \
0     1.5005  0.0162 -0.0034  …  1.6765  14.9509  0.5005  0.0118  0.0035
```

```
1     1.4966 -0.0005 -0.0148  …  1.1065  10.9003  0.5019  0.0223  0.0055
2     1.4436  0.0041  0.0013  …  2.0952   9.2721  0.4958  0.0157  0.0039
3     1.4882 -0.0124 -0.0033  …  1.7585   8.5831  0.4990  0.0103  0.0025
4     1.5031 -0.0031 -0.0072  …  1.6597  10.9698  0.4800  0.4766  0.1045
…       …       …       …   …      …       …       …       …       …
1562  1.3424 -0.0045 -0.0057  …  1.4879  11.7256  0.4988  0.0143  0.0039
1563  1.4333 -0.0061 -0.0093  …  1.0187  17.8379  0.4975  0.0131  0.0036
1564     NaN     NaN     NaN  …  1.2237  17.7267  0.4987  0.0153  0.0041
1565  1.4622 -0.0072  0.0032  …  1.7085  19.2104  0.5004  0.0178  0.0038
1566     NaN     NaN     NaN  …  1.2878  22.9183  0.4987  0.0181  0.0040

          432     433     434     435       436
0      2.3630     NaN     NaN     NaN       NaN
1      4.4447  0.0096  0.0201  0.0060  208.2045
2      3.1745  0.0584  0.0484  0.0148   82.8602
3      2.0544  0.0202  0.0149  0.0044   73.8432
4     99.3032  0.0202  0.0149  0.0044   73.8432
…         …       …       …       …        …
1562   2.8669  0.0068  0.0138  0.0047  203.1720
1563   2.6238  0.0068  0.0138  0.0047  203.1720
1564   3.0590  0.0197  0.0086  0.0025   43.5231
1565   3.5662  0.0262  0.0245  0.0075   93.4941
1566   3.6275  0.0117  0.0162  0.0045  137.7844

[1567 rows x 437 columns]
```

1.8 A Quick Check of the Output Data

[194]:
```
pd.DataFrame(y)
```

[194]:
```
         0
0     Pass
1     Pass
2     Fail
3     Pass
4     Pass
…     …
1562  Pass
1563  Pass
1564  Pass
1565  Pass
1566  Pass

[1567 rows x 1 columns]
```

1.9 Taking care of missing data

[195]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(X)
X = imputer.transform(X)
```

[196]:
```
# A quick check
print(X)
```

```
[[3.03093000e+03 2.56400000e+03 2.18773330e+03 … 1.64749042e-02
  5.28333333e-03 9.96700663e+01]
 [3.09578000e+03 2.46514000e+03 2.23042220e+03 … 2.01000000e-02
  6.00000000e-03 2.08204500e+02]
 [2.93261000e+03 2.55994000e+03 2.18641110e+03 … 4.84000000e-02
  1.48000000e-02 8.28602000e+01]
 …
 [2.97881000e+03 2.37978000e+03 2.20630000e+03 … 8.60000000e-03
  2.50000000e-03 4.35231000e+01]
 [2.89492000e+03 2.53201000e+03 2.17703330e+03 … 2.45000000e-02
  7.50000000e-03 9.34941000e+01]
 [2.94492000e+03 2.45076000e+03 2.19544440e+03 … 1.62000000e-02
  4.50000000e-03 1.37784400e+02]]
```

1.10 Encoding Categorical Data

1.10.1 Encoding the Independent Variable

[197]:
```
# we don't have any categorical data
```

1.10.2 Encoding the Dependent Variable

[198]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

[199]:
```
# a qucik check
print(y)
```

```
[1 1 0 … 1 1 1]
```

1.11 Feature Scaling

[200]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
```

1.12 Splitting the Dataset into the Training set and Test set

[201]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 34)
```

[202]:
```
print(X_train)
```

```
[[-0.96046311 -0.73813734 -0.92237938 … -0.06531583 -0.16868853
  -0.2120265 ]
 [-0.87742151  0.5426257  -0.13250295 …  0.60499301  0.38972867
   3.17408017]
 [ 0.05645609 -1.51130825  1.47184855 …  0.04829584  0.04071792
   0.42803032]
 …
 [-0.55464836 -0.10473817 -1.25868434 …  1.9342495   2.16968349
   0.21655552]
 [-0.24467179 -0.00336937 -1.42475658 … -0.64473532 -0.79690788
  -0.64548212]
 [-0.36283589 -0.07880372  0.55448143 … -0.19028866 -0.02908423
   1.62346601]]
```

[203]:
```
print(X_test)
```

```
[[ 0.38726113 -0.80185131  0.15916359 … -0.66745766 -0.65730358
  -0.56211928]
 [-0.02413509 -0.94137368 -0.96134565 … -1.08782082 -0.86671003
  -0.69795783]
 [ 0.70418053  0.53813704 -0.68064891 … -0.06531583  0.00581685
  -0.30543175]
 …
 [ 0.08504418  0.83962507 -1.23522968 … -1.12190432 -1.07611647
  -0.7688942 ]
 [ 2.27530028  0.21096393 -0.97685385 … -0.59929066 -0.55260035
  -0.6417691 ]
 [-0.8523729  -0.23341296 -0.28457524 … -0.50840133 -0.51769928
  -0.33965675]]
```

[204]:
```
print(y_train)
```

```
[0 1 1 … 1 1 1]
```

[205]:
```
print(y_test)
```

```
[1 1 1 1 1 0 1 1 1 1 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 0 1
 1 1 1 1 1 1 1 1 1 0 1 1 1 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 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 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 1 1 1 1 1 1 1 1 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 0 1 1 1 1 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
```

```
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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 1 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 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0]
```

[206]:
```python
print(X_train)
```

```
[[-0.96046311 -0.73813734 -0.92237938 … -0.06531583 -0.16868853
  -0.2120265 ]
 [-0.87742151  0.5426257  -0.13250295 …  0.60499301  0.38972867
   3.17408017]
 [ 0.05645609 -1.51130825  1.47184855 …  0.04829584  0.04071792
   0.42803032]
 …
 [-0.55464836 -0.10473817 -1.25868434 …  1.9342495   2.16968349
   0.21655552]
 [-0.24467179 -0.00336937 -1.42475658 … -0.64473532 -0.79690788
  -0.64548212]
 [-0.36283589 -0.07880372  0.55448143 … -0.19028866 -0.02908423
   1.62346601]]
```

[207]:
```python
print(X_test)
```

```
[[ 0.38726113 -0.80185131  0.15916359 … -0.66745766 -0.65730358
  -0.56211928]
 [-0.02413509 -0.94137368 -0.96134565 … -1.08782082 -0.86671003
  -0.69795783]
 [ 0.70418053  0.53813704 -0.68064891 … -0.06531583  0.00581685
  -0.30543175]
 …
 [ 0.08504418  0.83962507 -1.23522968 … -1.12190432 -1.07611647
  -0.7688942 ]
 [ 2.27530028  0.21096393 -0.97685385 … -0.59929066 -0.55260035
  -0.6417691 ]
 [-0.8523729  -0.23341296 -0.28457524 … -0.50840133 -0.51769928
  -0.33965675]]
```

2 Classification Models

We will investigate a logistic classification model, a Random Forest model and an SVM model. The script below will train each model and output comprehensive data about the performance.

[208]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
from sklearn.model_selection import cross_val_score, KFold
import seaborn as sns
```

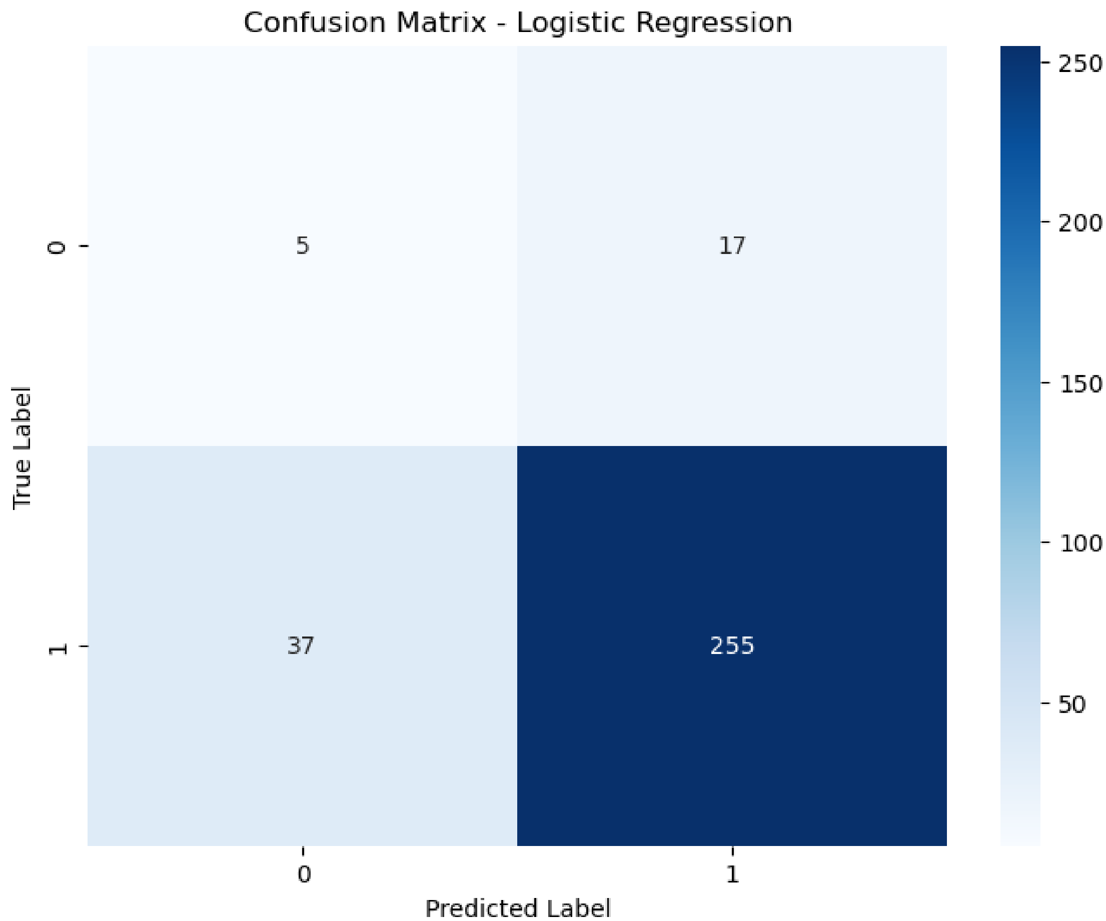
```python
def train_and_evaluate_model(model, X_train, X_test, y_train, y_test, model_name, needs_scaling=False):
    """
    Train and evaluate a single model, including k-fold validation
    """
    # Scale the features if needed (important for SVM)
    if needs_scaling:
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)
    else:
        X_train_scaled = X_train
        X_test_scaled = X_test

    # Train the model
    model.fit(X_train_scaled, y_train)

    # Make predictions
    y_pred = model.predict(X_test_scaled)
    y_pred_proba = model.predict_proba(X_test_scaled)

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    class_report = classification_report(y_test, y_pred, zero_division=0)
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Perform k-fold cross validation
    kf = KFold(n_splits=5, shuffle=True, random_state=42)
    cv_scores = cross_val_score(model, X_train_scaled, y_train, cv=kf, scoring='accuracy')

    # Print results
    print(f"\n{model_name} Results:")
    print("-" * 50)
    print(f"Test Set Accuracy: {accuracy:.4f}")
    print(f"\nK-fold Cross Validation Scores:")
    print(f"Mean CV Accuracy: {cv_scores.mean():.4f} (+/- {cv_scores.std() * 2:.4f})")
    print(f"Individual fold scores: {[f'{score:.4f}' for score in cv_scores]}")
    print("\nClassification Report:")
    print(class_report)

    # Plot confusion matrix
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix - {model_name}')
```

```python
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

    return {
        'accuracy': accuracy,
        'cv_scores': cv_scores,
        'conf_matrix': conf_matrix,
        'predictions': y_pred,
        'probabilities': y_pred_proba
    }

def compare_models(models_results):
    """
    Create comparison visualizations for the models
    """
    # Prepare data for box plot
    model_names = list(models_results.keys())
    cv_scores = [results['cv_scores'] for results in models_results.values()]
    test_accuracies = [results['accuracy'] for results in models_results.
 ↪values()]

    # Create box plot
    plt.figure(figsize=(12, 6))

    # Plot CV scores
    plt.subplot(1, 2, 1)
    plt.boxplot(cv_scores, labels=model_names)
    plt.title('Cross Validation Scores Comparison')
    plt.ylabel('Accuracy')
    plt.xticks(rotation=45)

    # Add individual points for CV scores
    for i in range(len(model_names)):
        x = np.random.normal(i + 1, 0.04, size=len(cv_scores[i]))
        plt.plot(x, cv_scores[i], 'r.', alpha=0.5)

    # Plot test accuracies
    plt.subplot(1, 2, 2)
    plt.bar(model_names, test_accuracies)
    plt.title('Test Set Accuracy Comparison')
    plt.ylabel('Accuracy')
    plt.xticks(rotation=45)

    plt.tight_layout()
    plt.show()
```

```python
# Initialize models
log_reg = LogisticRegression(random_state=35, max_iter=1000, class_weight='balanced',
                             solver='lbfgs')
rf_clf = RandomForestClassifier(random_state=35, class_weight='balanced',
                                n_estimators=100, max_depth=10)
svm_clf = SVC(kernel='rbf',  # RBF kernel for non-linear classification
              C=1.0,         # Regularization parameter
              gamma='scale', # Kernel coefficient
              class_weight='balanced',
              probability=True,
              random_state=35)

# Dictionary to store results
models_results = {}

# Train and evaluate all models
models_results['Logistic Regression'] = train_and_evaluate_model(
    log_reg, X_train, X_test, y_train, y_test, "Logistic Regression"
)

models_results['Random Forest'] = train_and_evaluate_model(
    rf_clf, X_train, X_test, y_train, y_test, "Random Forest"
)

models_results['SVM'] = train_and_evaluate_model(
    svm_clf, X_train, X_test, y_train, y_test, "SVM", needs_scaling=True  # SVM needs scaled features
)

# Compare models
compare_models(models_results)

# Feature importance for Random Forest and coefficients for Logistic Regression
if hasattr(X_train, 'columns'):
    # Random Forest feature importance
    rf_importance = pd.DataFrame({
        'feature': X_train.columns,
        'importance': rf_clf.feature_importances_
    }).sort_values('importance', ascending=False)

    # Logistic Regression coefficients (absolute values)
    lr_importance = pd.DataFrame({
        'feature': X_train.columns,
        'importance': np.abs(log_reg.coef_[0])
    }).sort_values('importance', ascending=False)
```

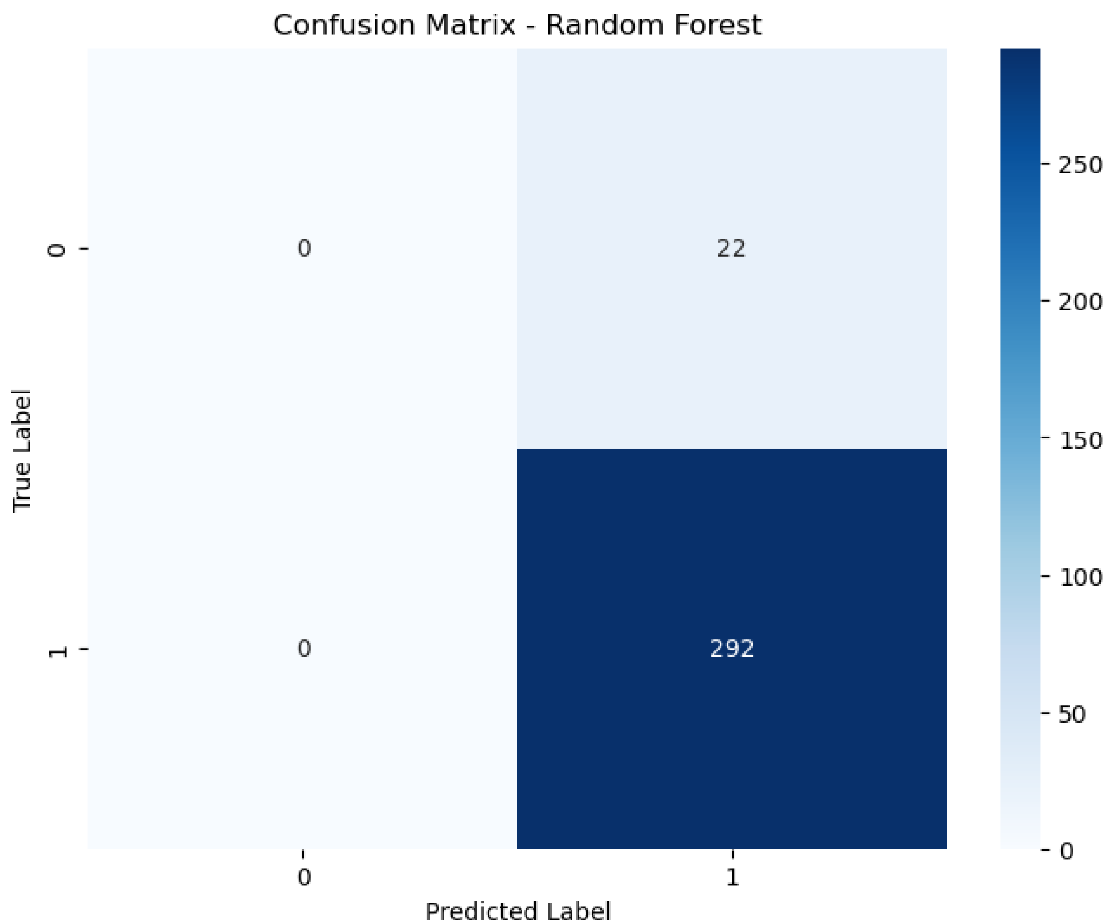
```
    # Plot feature importance
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    sns.barplot(data=rf_importance, x='importance', y='feature')
    plt.title('Random Forest Feature Importance')

    plt.subplot(1, 2, 2)
    sns.barplot(data=lr_importance, x='importance', y='feature')
    plt.title('Logistic Regression Feature Importance')

    plt.tight_layout()
    plt.show()

# Calculate and display model comparison statistics
comparison_stats = pd.DataFrame({
    'Model': models_results.keys(),
    'Test Accuracy': [results['accuracy'] for results in models_results.
 ↪values()],
    'CV Mean Accuracy': [results['cv_scores'].mean() for results in␣
 ↪models_results.values()],
    'CV Std': [results['cv_scores'].std() for results in models_results.
 ↪values()]
})

print("\nModel Comparison Summary:")
print("-" * 80)
print(comparison_stats.to_string(index=False))
```

```
Logistic Regression Results:
--------------------------------------------------
Test Set Accuracy: 0.8280

K-fold Cross Validation Scores:
Mean CV Accuracy: 0.8412 (+/- 0.0372)
Individual fold scores: ['0.8167', '0.8247', '0.8526', '0.8680', '0.8440']

Classification Report:
              precision    recall  f1-score   support

           0       0.12      0.23      0.16        22
           1       0.94      0.87      0.90       292

    accuracy                           0.83       314
   macro avg       0.53      0.55      0.53       314
weighted avg       0.88      0.83      0.85       314
```



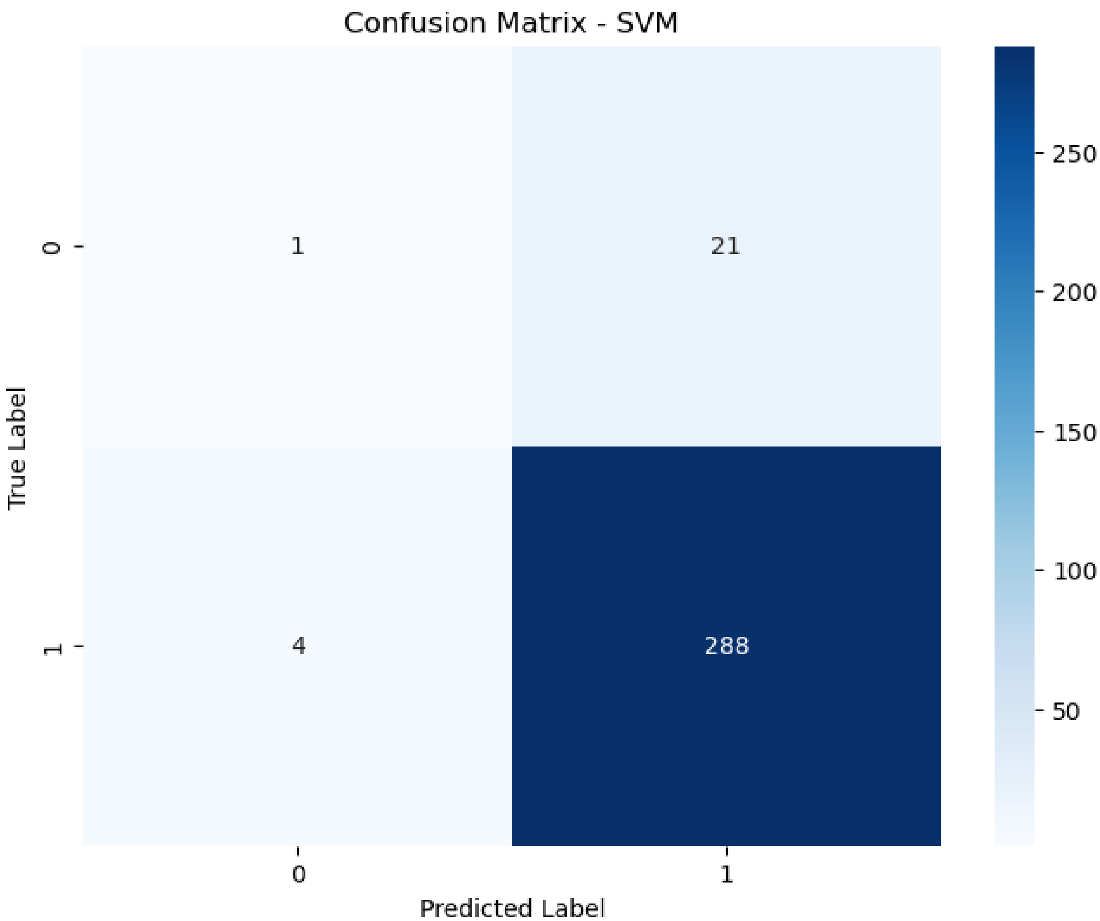

```
Random Forest Results:
--------------------------------------------------
Test Set Accuracy: 0.9299

K-fold Cross Validation Scores:
Mean CV Accuracy: 0.9345 (+/- 0.0374)
Individual fold scores: ['0.9163', '0.9522', '0.9522', '0.9080', '0.9440']

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        22
           1       0.93      1.00      0.96       292

    accuracy                           0.93       314
   macro avg       0.46      0.50      0.48       314
```

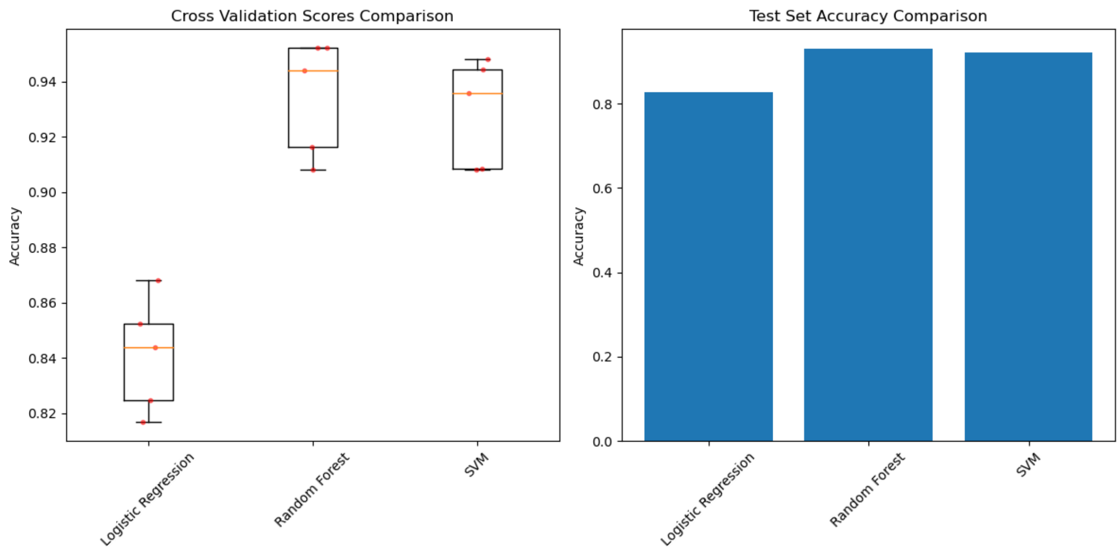
```
weighted avg       0.86      0.93      0.90       314
```




```
SVM Results:
--------------------------------------------------
Test Set Accuracy: 0.9204

K-fold Cross Validation Scores:
Mean CV Accuracy: 0.9290 (+/- 0.0348)
Individual fold scores: ['0.9084', '0.9442', '0.9482', '0.9080', '0.9360']

Classification Report:
              precision    recall  f1-score   support

           0       0.20      0.05      0.07        22
           1       0.93      0.99      0.96       292

    accuracy                           0.92       314
```

```
   macro avg       0.57      0.52      0.52       314
weighted avg       0.88      0.92      0.90       314
```






```
Model Comparison Summary:
--------------------------------------------------------------------------------
               Model  Test Accuracy  CV Mean Accuracy    CV Std
Logistic Regression       0.828025          0.841205 0.018583
      Random Forest       0.929936          0.934543 0.018700
                SVM       0.920382          0.928959 0.017415
```

2.1 Artificial Neural Network

We will try to optimize the neural network parameters for the same problem. We will then evaluate how well the ANN model predicts the pass or fail criteria.

[209]:
```
import tensorflow as tf
from tensorflow import keras
from keras import callbacks
```

Since the Dataset has 436 input features, we will also look into whether apply PCA is beneficial for generalization of the model as well as to improve computing time.

1. Without PCA, feed all the 436 features into ANN model
2. Apply PCA, and optimize parameters for the ANN model

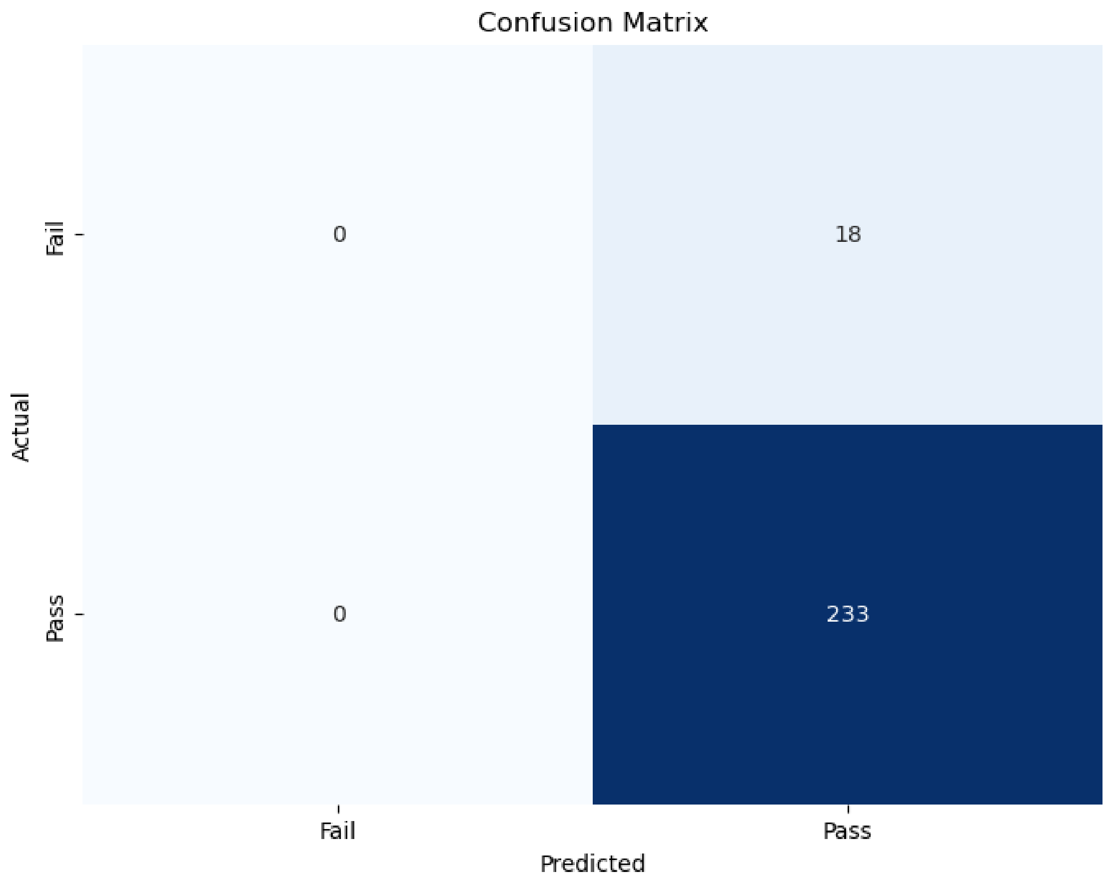
[210]:
```
import numpy as np
from sklearn.model_selection import KFold
from sklearn.decomposition import PCA

def create_model():
    # Initialize the ANN model
    model = tf.keras.models.Sequential()
```

```
    # Input Layer
    model.add(tf.keras.layers.Dense(units=436, activation='relu'))

    # Hidden Layers with Dropout and Regularization
    model.add(tf.keras.layers.Dense(700, activation='relu',␣
↪kernel_regularizer=tf.keras.regularizers.l2(0.01)))
    model.add(tf.keras.layers.Dropout(0.2))
    model.add(tf.keras.layers.Dense(400, activation='relu',␣
↪kernel_regularizer=tf.keras.regularizers.l2(0.01)))
    model.add(tf.keras.layers.Dropout(0.3))

    # Output Layer
    model.add(tf.keras.layers.Dense(1, activation='sigmoid'))  # Change to␣
↪softmax for multi-class

    # Compile the model
    model.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['accuracy'])  # Change loss for multi-class
    return model

# Ensure X and y are numpy arrays
X_train = np.array(X_train)
y_train = np.array(y_train)

# # Fit PCA
# pca = PCA(n_components=0.95)  # Retain 95% of the variance
# X_pca = pca.fit_transform(X)

# Initialize K-fold cross-validation
n_splits = 5
kfold = KFold(n_splits=n_splits, shuffle=True, random_state=42)

# Lists to store metrics
fold_accuracies = []
fold_losses = []
histories = []
all_predictions = []
all_true_values = []

# K-fold cross validation
for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train)):
    print(f'\nFold {fold + 1}/{n_splits}')

    # Split data
    X_train_fold = X_train[train_idx]
    y_train_fold = y[train_idx]
```

```
    X_val_fold = X_train[val_idx]
    y_val_fold = y[val_idx]

    # Create and compile model
    model = create_model()

    # Callbacks
    lr_scheduler = callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=5
    )
    early_stopping = callbacks.EarlyStopping(
        monitor='val_loss',
        patience=10,
        restore_best_weights=True
    )

    # Train model
    history = model.fit(
        X_train_fold,
        y_train_fold,
        batch_size=256,
        epochs=100,
        validation_data=(X_val_fold, y_val_fold),
        callbacks=[lr_scheduler, early_stopping],
        verbose=0
    )

    # Evaluate model
    loss, accuracy = model.evaluate(X_val_fold, y_val_fold, verbose=0)
    fold_accuracies.append(accuracy)
    fold_losses.append(loss)
    histories.append(history.history)

    # Generate predictions
    y_pred = model.predict(X_val_fold, verbose=0)
    y_pred_binary = (y_pred > 0.5).astype(int).reshape(-1)  # Ensure 1D array

    # Store predictions and true values
    all_predictions.extend(y_pred_binary)
    all_true_values.extend(y_val_fold)

    # Calculate metrics for this fold
    conf_matrix = confusion_matrix(y_val_fold, y_pred_binary)

    print(f'\nFold {fold + 1} Results:')
```

```
    print(f'Validation Loss: {loss:.4f}')
    print(f'Validation Accuracy: {accuracy:.4f}')
    print('\nConfusion Matrix:')

    # Plot Confusion Matrix
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
                xticklabels=['Fail', 'Pass'], yticklabels=['Fail', 'Pass'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()

    print('\nClassification Report:')
    print(classification_report(y_val_fold, y_pred_binary,␣
 ↪target_names=['Fail', 'Pass']))

# Convert to numpy arrays for final calculations
all_predictions = np.array(all_predictions)
all_true_values = np.array(all_true_values)

# Print overall results
print('\nOverall K-fold Cross Validation Results:')
print('----------------------------------------')
print(f'Mean Accuracy: {np.mean(fold_accuracies):.4f} (±{np.
 ↪std(fold_accuracies):.4f})')
print(f'Mean Loss: {np.mean(fold_losses):.4f} (±{np.std(fold_losses):.4f})')

# Overall confusion matrix and classification report
print('\nOverall Confusion Matrix:')
print(confusion_matrix(all_true_values, all_predictions))
```

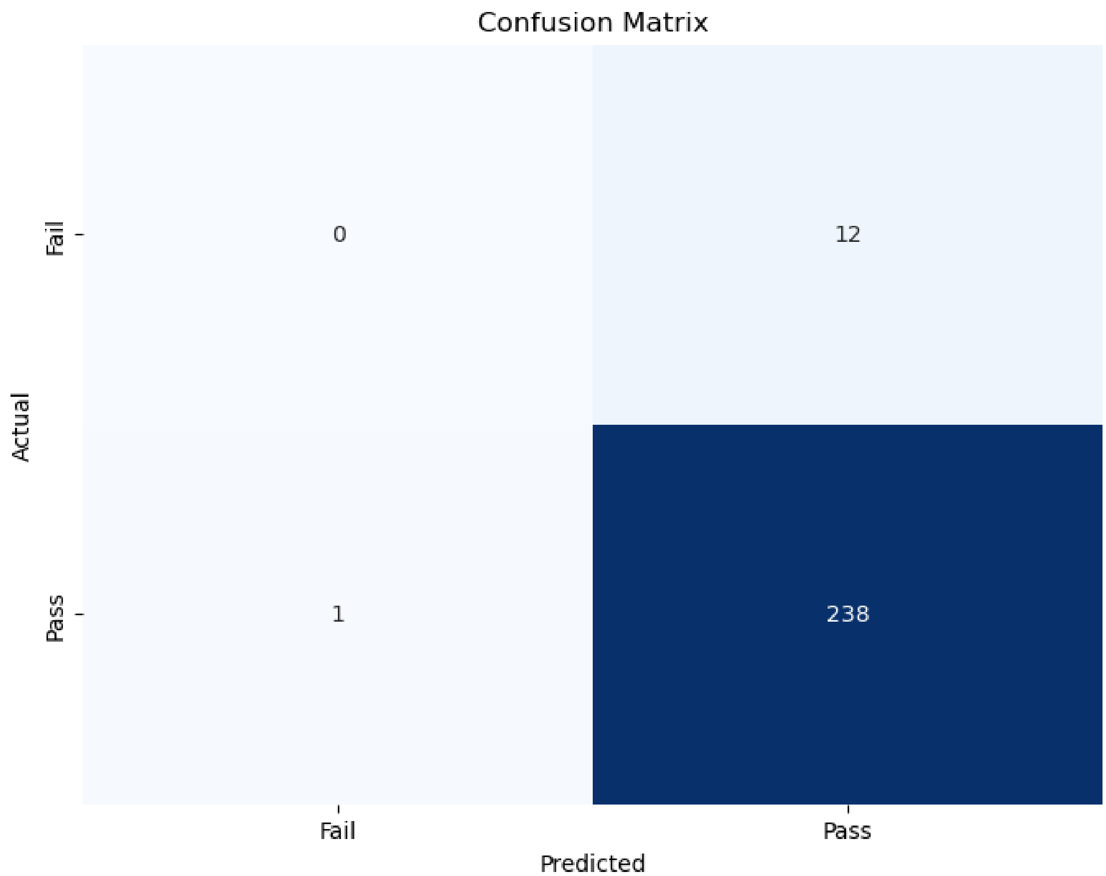
```
Fold 1/5

Fold 1 Results:
Validation Loss: 0.4714
Validation Accuracy: 0.9283

Confusion Matrix:
```




```
Classification Report:
              precision    recall  f1-score   support

        Fail       0.00      0.00      0.00        18
        Pass       0.93      1.00      0.96       233

    accuracy                           0.93       251
   macro avg       0.46      0.50      0.48       251
weighted avg       0.86      0.93      0.89       251


Fold 2/5
```

```
c:\Users\kylea\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\kylea\anaconda3\lib\site-
```

```
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\kylea\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))


Fold 2 Results:
Validation Loss: 0.3533
Validation Accuracy: 0.9482

Confusion Matrix:
```



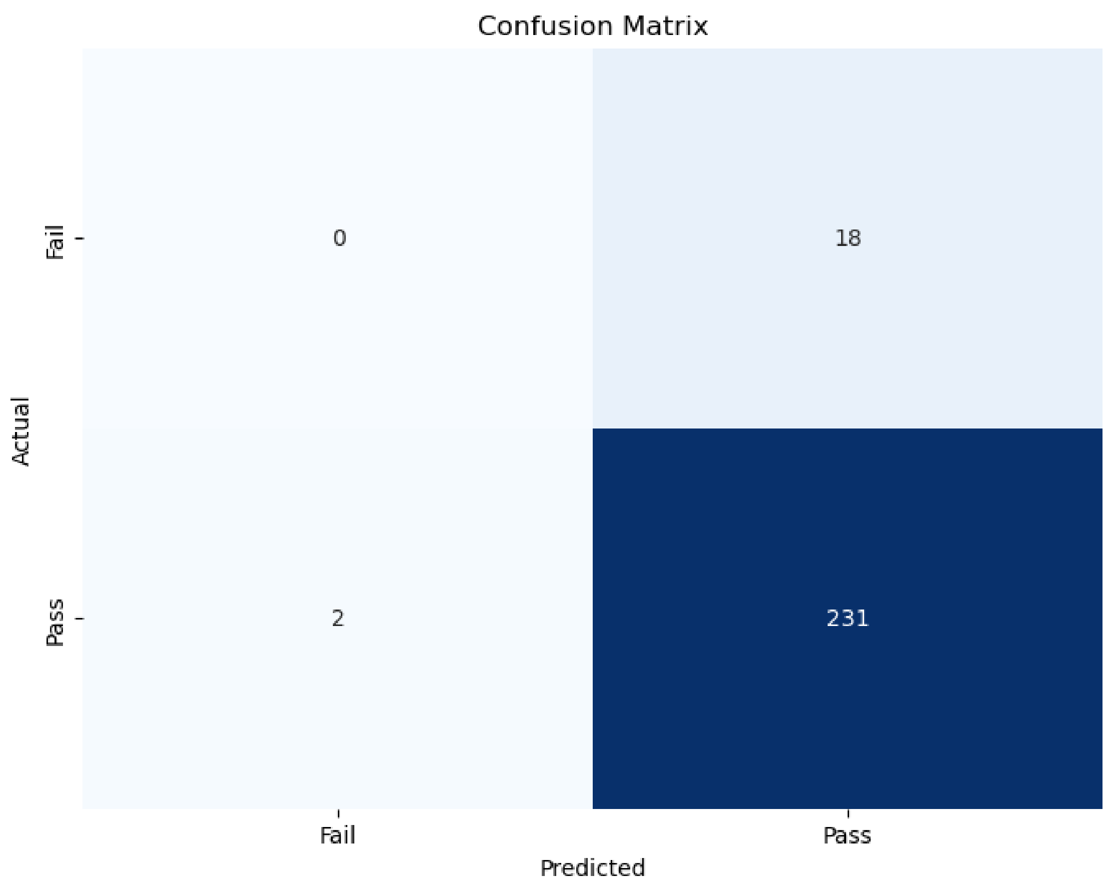

```
Classification Report:
              precision    recall  f1-score   support
```

	precision	recall	f1-score	support
Fail	0.00	0.00	0.00	12
Pass	0.95	1.00	0.97	239
accuracy			0.95	251
macro avg	0.48	0.50	0.49	251
weighted avg	0.91	0.95	0.93	251

Fold 3/5

Fold 3 Results:
Validation Loss: 0.5179
Validation Accuracy: 0.9203

Confusion Matrix:



Classification Report:

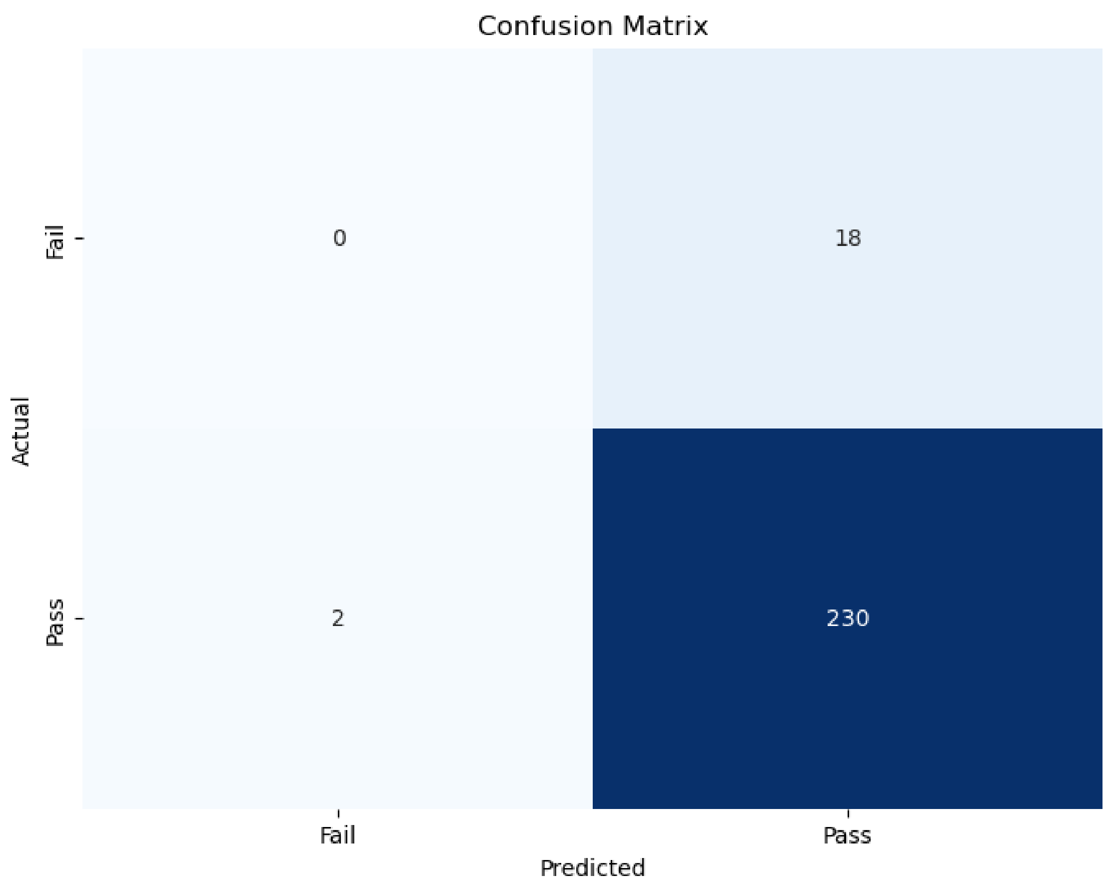
precision recall f1-score support

	precision	recall	f1-score	support
Fail	0.00	0.00	0.00	18
Pass	0.93	0.99	0.96	233
accuracy			0.92	251
macro avg	0.46	0.50	0.48	251
weighted avg	0.86	0.92	0.89	251

Fold 4/5

Fold 4 Results:
Validation Loss: 0.4936
Validation Accuracy: 0.9200

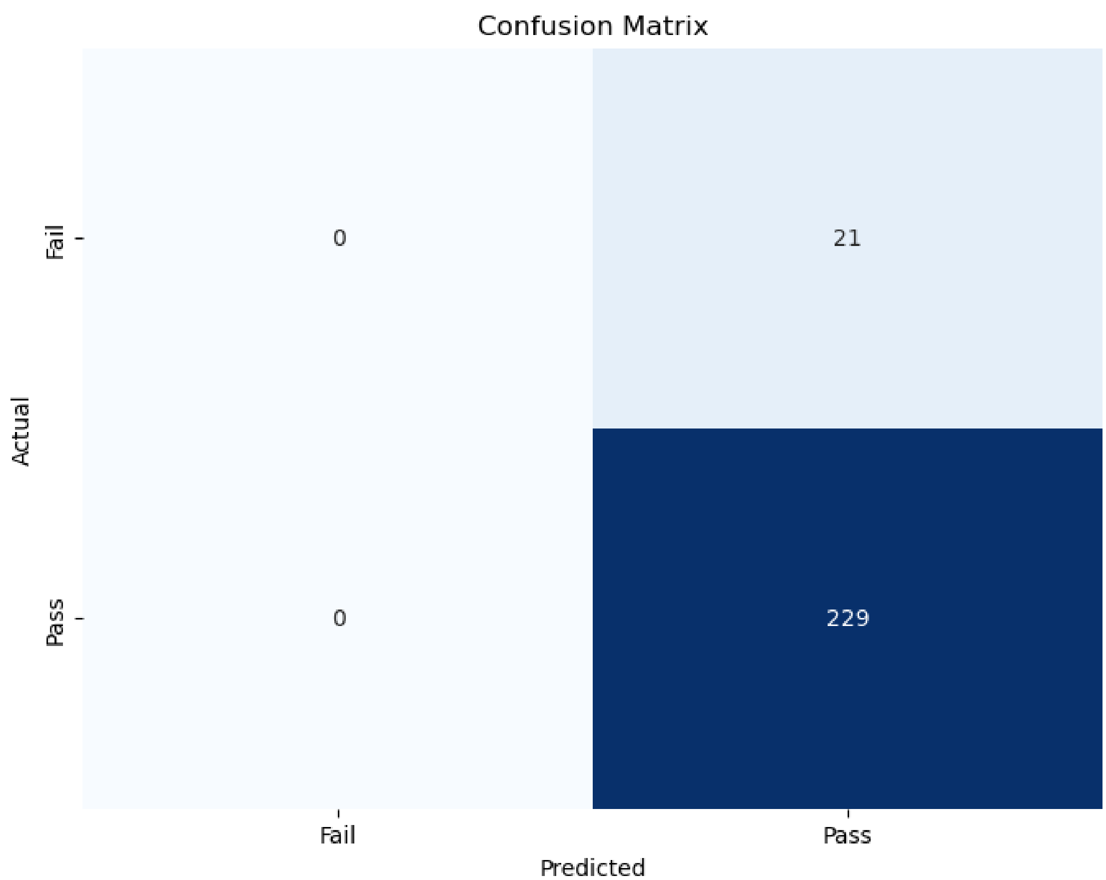
Confusion Matrix:



Classification Report:

```
              precision    recall  f1-score   support
```

```
        Fail       0.00      0.00      0.00        18
        Pass       0.93      0.99      0.96       232

    accuracy                           0.92       250
   macro avg       0.46      0.50      0.48       250
weighted avg       0.86      0.92      0.89       250


Fold 5/5

Fold 5 Results:
Validation Loss: 0.6383
Validation Accuracy: 0.9160

Confusion Matrix:
```




```
Classification Report:
              precision    recall  f1-score   support
```

```
        Fail       0.00      0.00      0.00        21
        Pass       0.92      1.00      0.96       229

    accuracy                           0.92       250
   macro avg       0.46      0.50      0.48       250
weighted avg       0.84      0.92      0.88       250


Overall K-fold Cross Validation Results:
----------------------------------------
Mean Accuracy: 0.9266 (±0.0115)
Mean Loss: 0.4949 (±0.0913)

Overall Confusion Matrix:
[[   0   87]
 [   5 1161]]
```

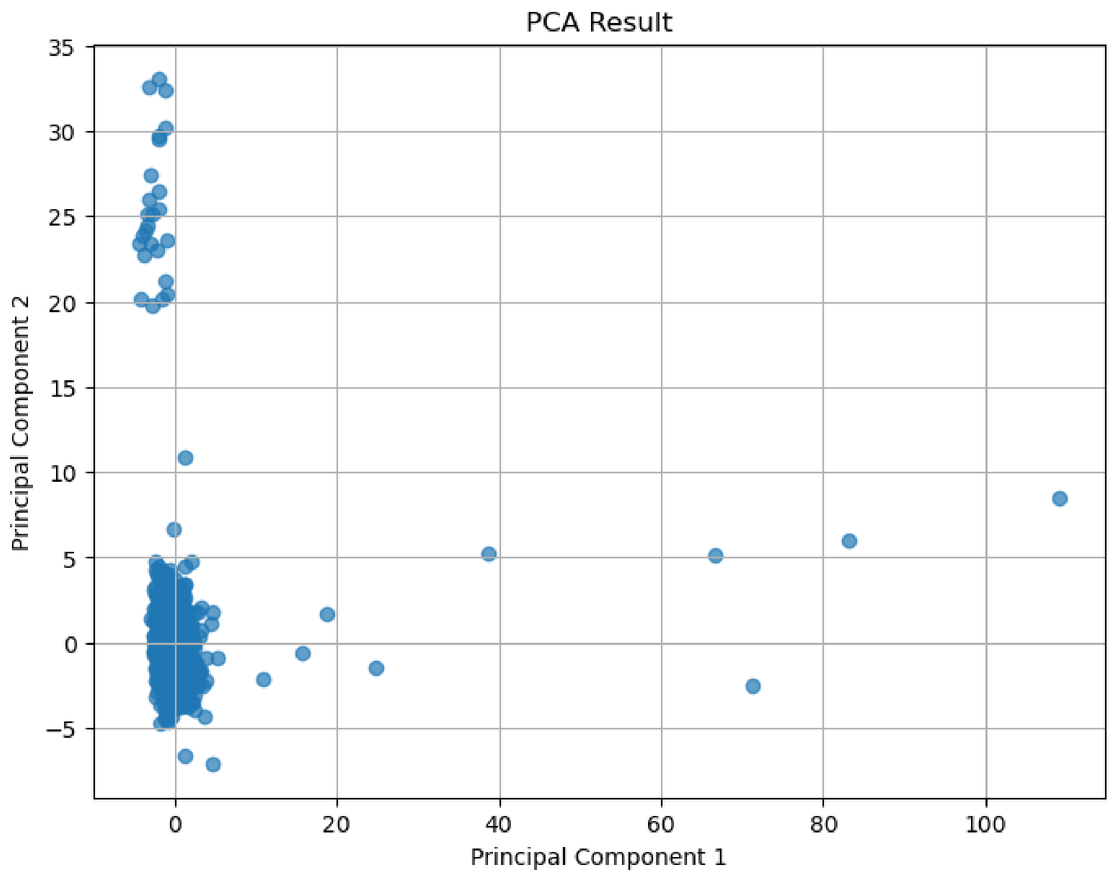
```
c:\Users\kylea\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\kylea\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\kylea\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

2.2 Implement PCA - ANN

[211]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# You can also concatenate your train and test data if needed for PCA fitting
# X = np.vstack((X_train, X_test))

# Standardize the data (important for PCA)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train)

# Initialize PCA
pca = PCA(n_components=2)  # Adjust the number of components as needed
```

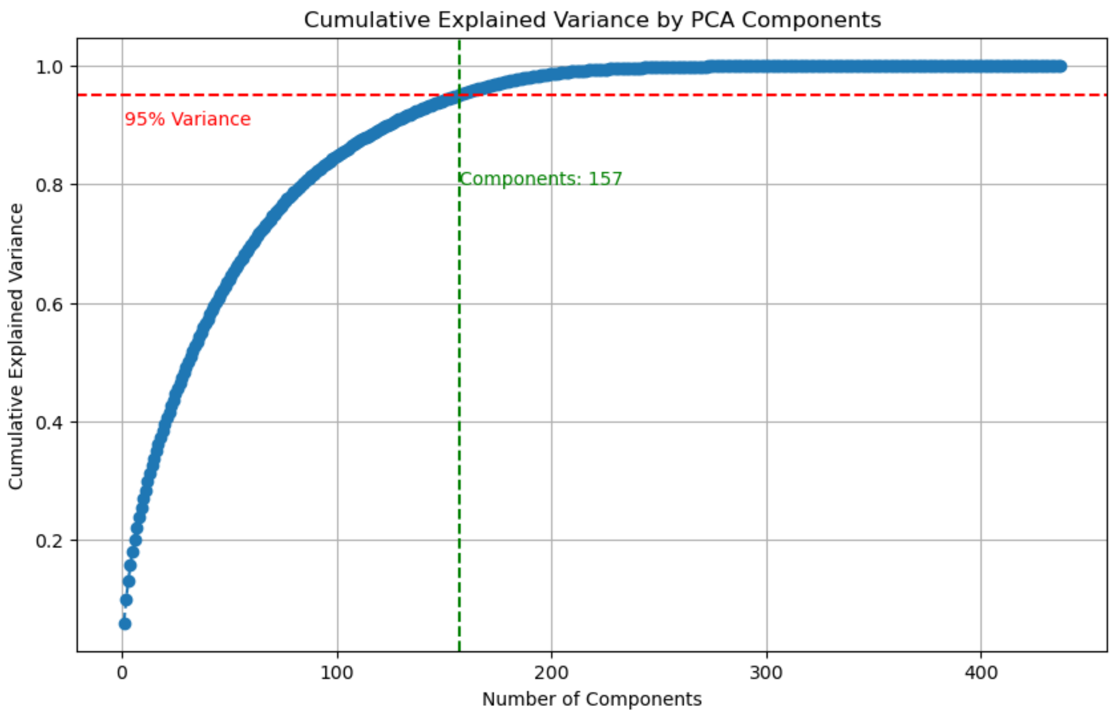
```python
# Fit PCA on the scaled data
X_pca = pca.fit_transform(X_scaled)

# Create a DataFrame for PCA results
pca_df = pd.DataFrame(data=X_pca, columns=['Principal Component 1', 'Principal␣
↪Component 2'])

# Explained variance ratio
explained_variance = pca.explained_variance_ratio_
print(f'Explained variance ratio: {explained_variance}')

# Plotting the PCA results
plt.figure(figsize=(8, 6))
plt.scatter(pca_df['Principal Component 1'], pca_df['Principal Component 2'],␣
↪alpha=0.7)
plt.title('PCA Result')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid()
plt.show()
```

```
Explained variance ratio: [0.06044871 0.03900913]
```



[212]:
```
# Fit PCA again to get explained variance ratio
pca = PCA()
pca.fit(X_scaled)

# Plot the explained variance
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(pca.explained_variance_ratio_) + 1),
         np.cumsum(pca.explained_variance_ratio_),
         marker='o', linestyle='--')
plt.title('Cumulative Explained Variance by PCA Components')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.axhline(y=0.95, color='r', linestyle='--')  # Line for 95% variance
plt.axvline(x=np.argmax(np.cumsum(pca.explained_variance_ratio_) >= 0.95) + 1,
  color='g', linestyle='--')  # Line for the number of components needed for
  95%
plt.text(1, 0.9, '95% Variance', color='red')
```

```
plt.text(np.argmax(np.cumsum(pca.explained_variance_ratio_) >= 0.95) + 1, 0.8,␣
  ↪f'Components: {np.argmax(np.cumsum(pca.explained_variance_ratio_) >= 0.95) +␣
  ↪1}', color='green')
plt.grid()
plt.show()

# Print the number of components used in PCA
n_components = np.argmax(np.cumsum(pca.explained_variance_ratio_) >= 0.95) + 1
print(f"Number of components retained to achieve 95% variance: {n_components}")
```




```
Number of components retained to achieve 95% variance: 157
```

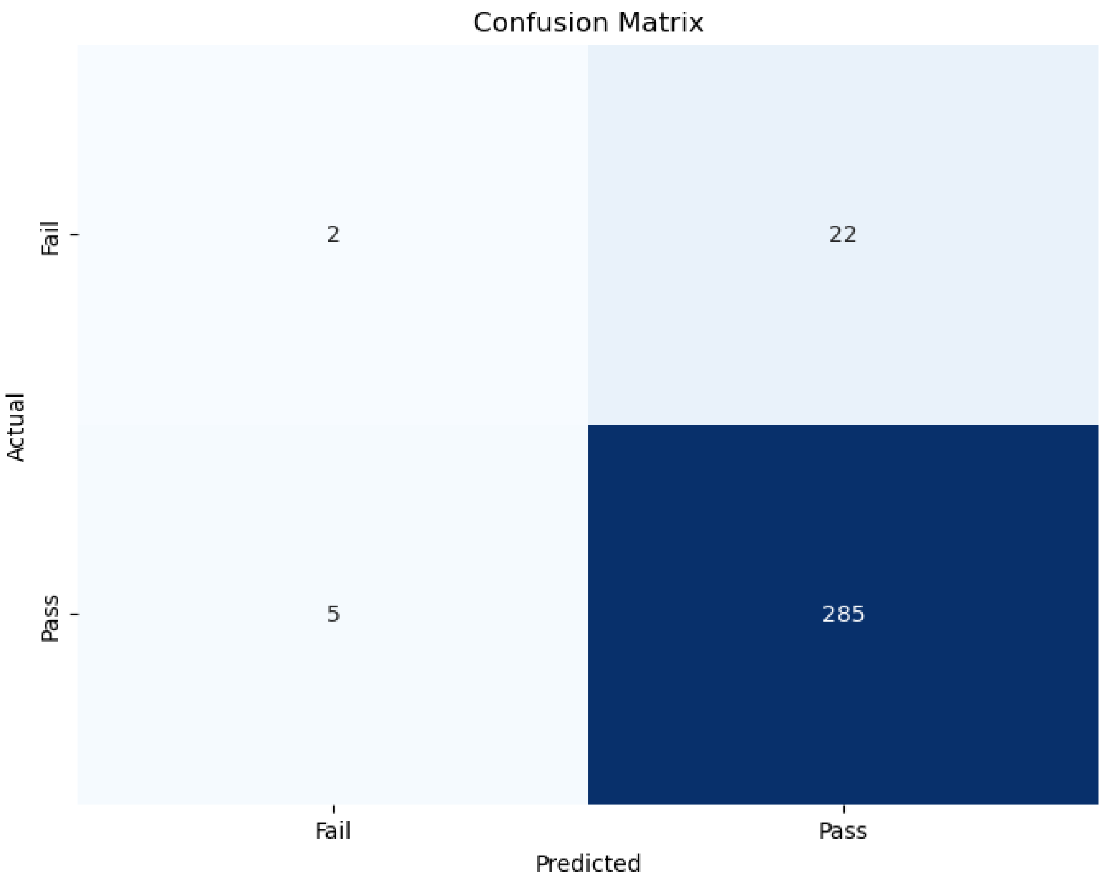
[213]:
```
import numpy as np
from sklearn.model_selection import KFold
from sklearn.decomposition import PCA

def create_model():
    # Initialize the ANN model
    model = tf.keras.models.Sequential()

    # Input Layer
    model.add(tf.keras.layers.Dense(units=157, activation='relu'))

    # Hidden Layers with Dropout
```

```
    model.add(tf.keras.layers.Dense(252, activation='relu',␣
↪kernel_regularizer=tf.keras.regularizers.l2(0.01)))
    model.add(tf.keras.layers.Dropout(0.2))
    model.add(tf.keras.layers.Dense(150, activation='relu',␣
↪kernel_regularizer=tf.keras.regularizers.l2(0.01)))
    model.add(tf.keras.layers.Dropout(0.3))

    # Output Layer
    model.add(tf.keras.layers.Dense(1, activation='sigmoid'))  # Change to␣
↪softmax for multi-class

    # Compile the model
    model.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['accuracy'])  # Change loss for multi-class
    return model

# Ensure X and y are numpy arrays
X = np.array(X)
y = np.array(y)

# Fit PCA
pca = PCA(n_components=0.95)  # Retain 95% of the variance
X_pca = pca.fit_transform(X)

# Initialize K-fold cross-validation
n_splits = 5
kfold = KFold(n_splits=n_splits, shuffle=True, random_state=42)

# Lists to store metrics
fold_accuracies = []
fold_losses = []
histories = []
all_predictions = []
all_true_values = []

# K-fold cross validation
for fold, (train_idx, val_idx) in enumerate(kfold.split(X_pca)):
    print(f'\nFold {fold + 1}/{n_splits}')

    # Split data
    X_train_fold = X_pca[train_idx]
    y_train_fold = y[train_idx]
    X_val_fold = X_pca[val_idx]
    y_val_fold = y[val_idx]

    # Create and compile model
    model = create_model()
```

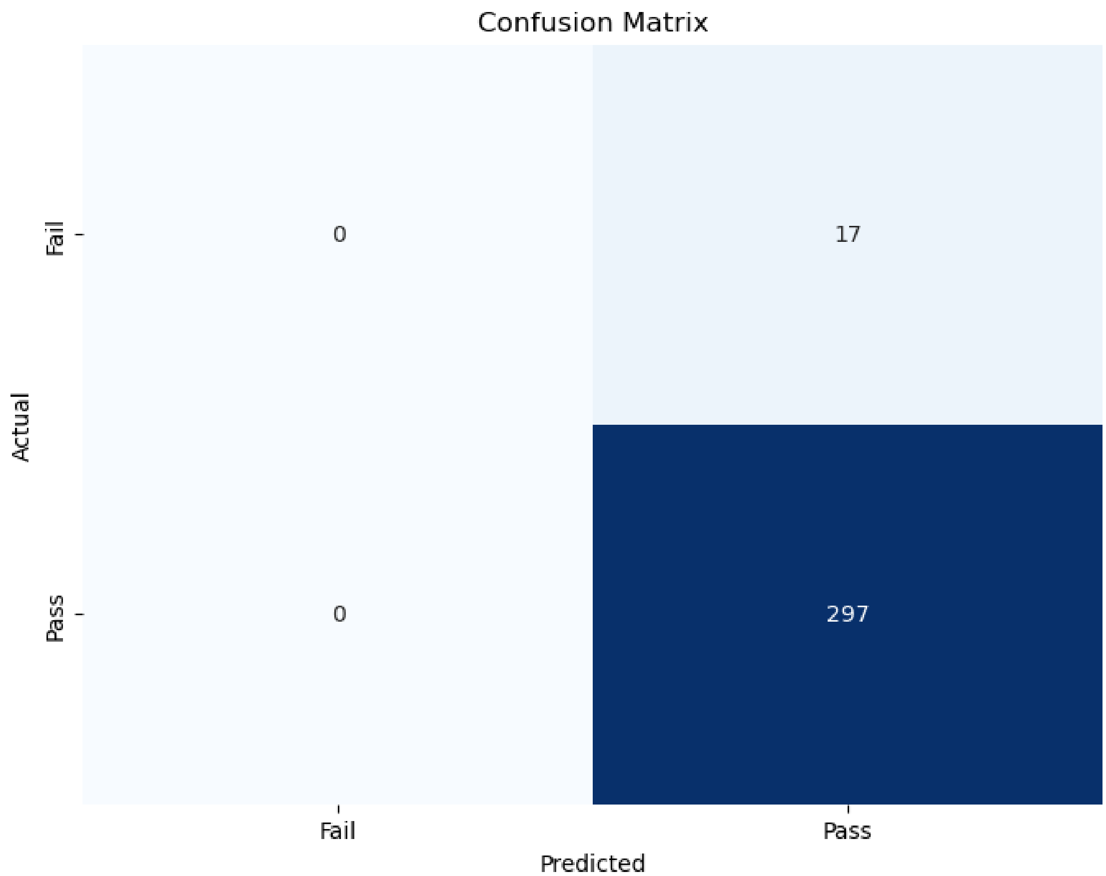
```
# Callbacks
lr_scheduler = callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=5
)
early_stopping = callbacks.EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)

# Train model
history = model.fit(
    X_train_fold,
    y_train_fold,
    batch_size=256,
    epochs=100,
    validation_data=(X_val_fold, y_val_fold),
    callbacks=[lr_scheduler, early_stopping],
    verbose=0
)

# Evaluate model
loss, accuracy = model.evaluate(X_val_fold, y_val_fold, verbose=0)
fold_accuracies.append(accuracy)
fold_losses.append(loss)
histories.append(history.history)

# Generate predictions
y_pred = model.predict(X_val_fold, verbose=0)
y_pred_binary = (y_pred > 0.5).astype(int).reshape(-1)  # Ensure 1D array

# Store predictions and true values
all_predictions.extend(y_pred_binary)
all_true_values.extend(y_val_fold)

# Calculate metrics for this fold
conf_matrix = confusion_matrix(y_val_fold, y_pred_binary)

print(f'\nFold {fold + 1} Results:')
print(f'Validation Loss: {loss:.4f}')
print(f'Validation Accuracy: {accuracy:.4f}')
print('\nConfusion Matrix:')

# Plot Confusion Matrix
```

```
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
                xticklabels=['Fail', 'Pass'], yticklabels=['Fail', 'Pass'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()

    print('\nClassification Report:')
    print(classification_report(y_val_fold, y_pred_binary,␣
 ↪target_names=['Fail', 'Pass']))

# Convert to numpy arrays for final calculations
all_predictions = np.array(all_predictions)
all_true_values = np.array(all_true_values)

# Print overall results
print('\nOverall K-fold Cross Validation Results:')
print('----------------------------------------')
print(f'Mean Accuracy: {np.mean(fold_accuracies):.4f} (±{np.
 ↪std(fold_accuracies):.4f})')
print(f'Mean Loss: {np.mean(fold_losses):.4f} (±{np.std(fold_losses):.4f})')

# Overall confusion matrix and classification report
print('\nOverall Confusion Matrix:')
print(confusion_matrix(all_true_values, all_predictions))
```

```
Fold 1/5

Fold 1 Results:
Validation Loss: 0.4326
Validation Accuracy: 0.9140

Confusion Matrix:
```



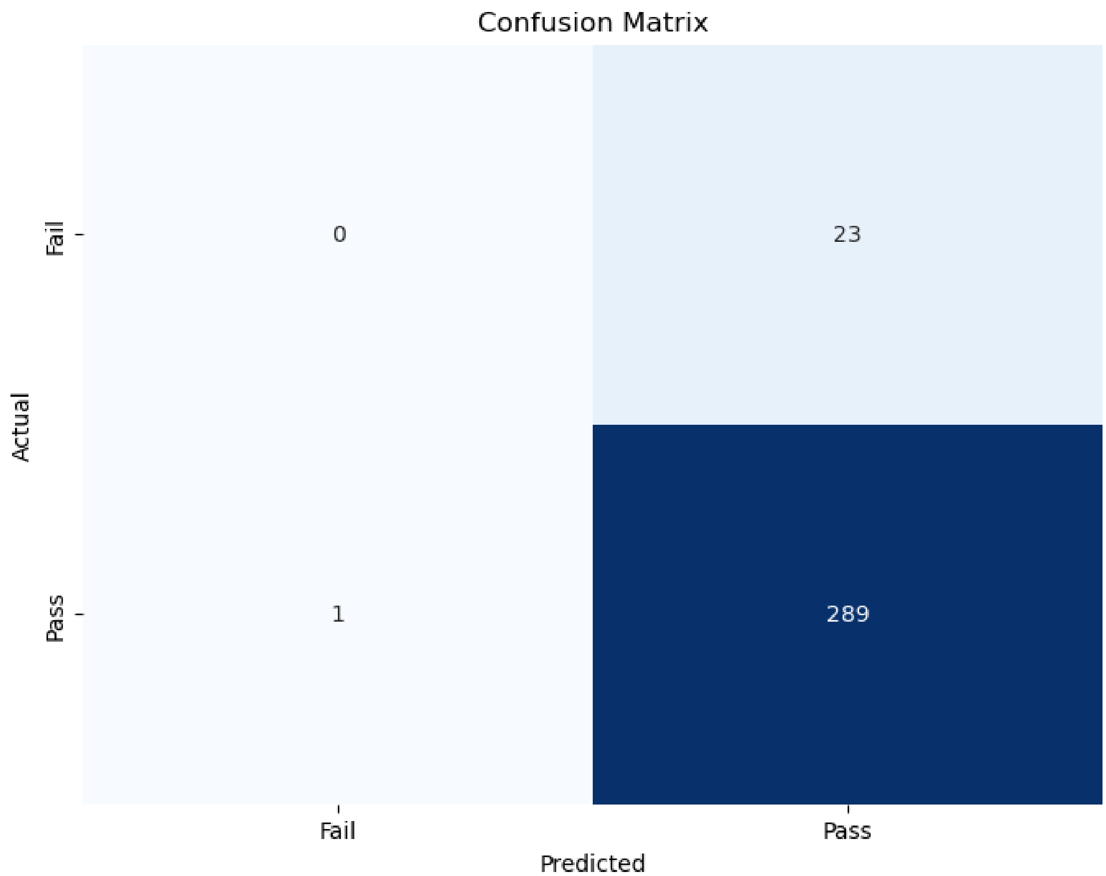

```
Classification Report:
              precision    recall  f1-score   support

        Fail       0.29      0.08      0.13        24
        Pass       0.93      0.98      0.95       290

    accuracy                           0.91       314
   macro avg       0.61      0.53      0.54       314
weighted avg       0.88      0.91      0.89       314


Fold 2/5

Fold 2 Results:
Validation Loss: 0.2815
Validation Accuracy: 0.9459

Confusion Matrix:
```




```
Classification Report:
              precision    recall  f1-score   support

        Fail       0.00      0.00      0.00        17
        Pass       0.95      1.00      0.97       297

    accuracy                           0.95       314
   macro avg       0.47      0.50      0.49       314
weighted avg       0.89      0.95      0.92       314


Fold 3/5
```

```
c:\Users\kylea\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\kylea\anaconda3\lib\site-
```

```
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\kylea\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))


Fold 3 Results:
Validation Loss: 0.4124
Validation Accuracy: 0.9233

Confusion Matrix:
```



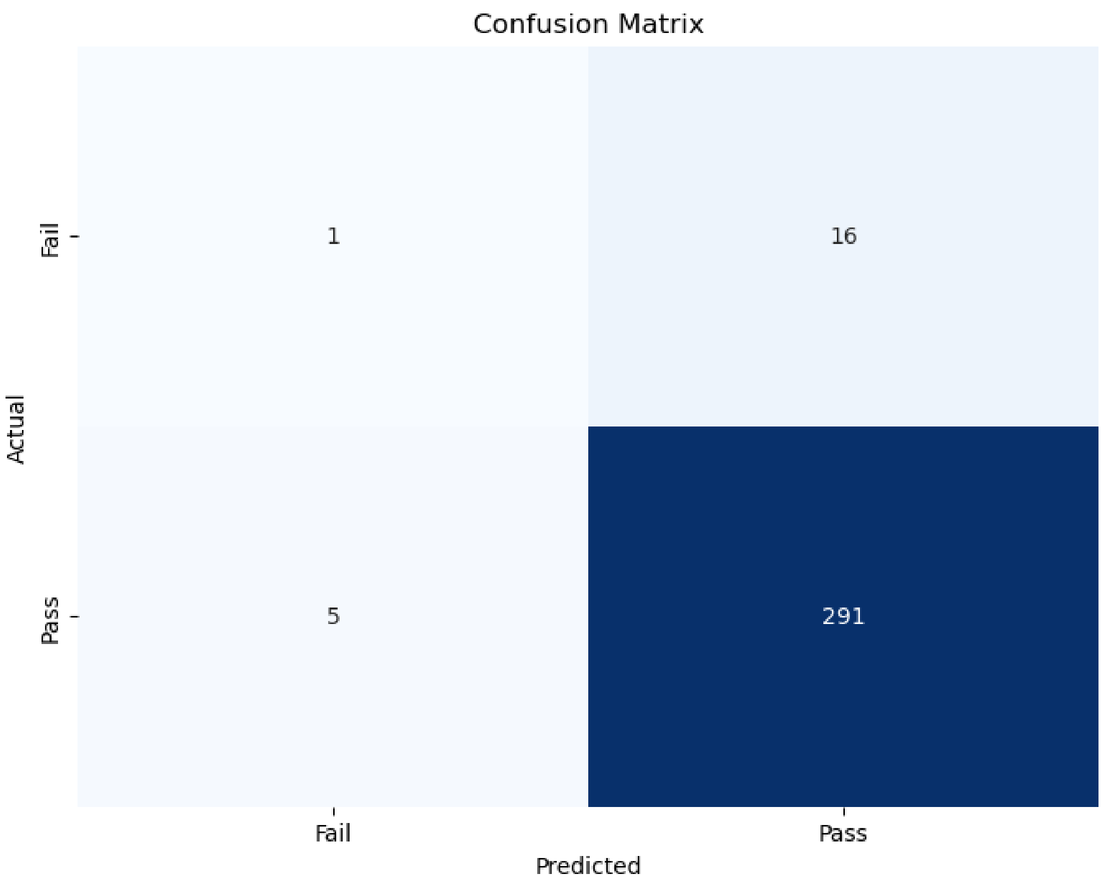

```
Classification Report:
              precision    recall  f1-score   support
```

	precision	recall	f1-score	support
Fail	0.00	0.00	0.00	23
Pass	0.93	1.00	0.96	290
accuracy			0.92	313
macro avg	0.46	0.50	0.48	313
weighted avg	0.86	0.92	0.89	313

Fold 4/5

Fold 4 Results:
Validation Loss: 0.3627
Validation Accuracy: 0.9329

Confusion Matrix:



Classification Report:

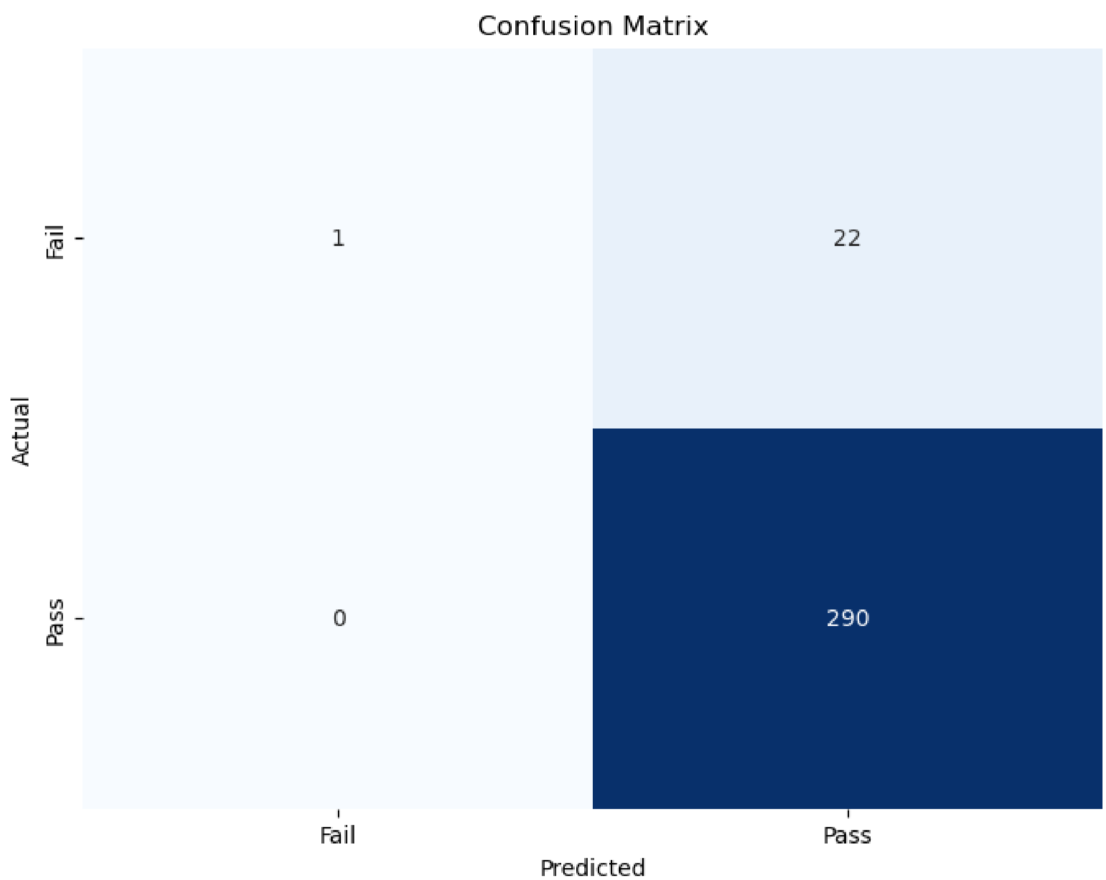
precision recall f1-score support

```
        Fail       0.17      0.06      0.09        17
        Pass       0.95      0.98      0.97       296

    accuracy                           0.93       313
   macro avg       0.56      0.52      0.53       313
weighted avg       0.91      0.93      0.92       313
```

Fold 5/5

Fold 5 Results:
Validation Loss: 0.4505
Validation Accuracy: 0.9297

Confusion Matrix:



Classification Report:

```
              precision    recall  f1-score   support
```

```
        Fail       1.00      0.04      0.08        23
        Pass       0.93      1.00      0.96       290

    accuracy                           0.93       313
   macro avg       0.96      0.52      0.52       313
weighted avg       0.93      0.93      0.90       313


Overall K-fold Cross Validation Results:
----------------------------------------
Mean Accuracy: 0.9292 (±0.0106)
Mean Loss: 0.3879 (±0.0608)

Overall Confusion Matrix:
[[   4  100]
 [  11 1452]]
```

3 Summary

4 Neural Network Architecture and Model Comparison Analysis

4.1 Model Architectures

4.1.1 ANN without PCA

- First Hidden Layer: 700 nodes
- Second Hidden Layer: 400 nodes
- Dropout Rate: 0.3
- Test Accuracy: 92.66%
- Loss Function: 0.50

4.1.2 ANN with PCA (95% variance retained)

- First Hidden Layer: 252 nodes
- Second Hidden Layer: 150 nodes
- Test Accuracy: 92.92%
- Loss Function: 0.39

4.1.3 Traditional Classification Models

Model	Test Accuracy	CV Mean Accuracy	CV Std
Logistic Regression	82.80%	84.13%	0.018583
Random Forest	92.99%	93.45%	0.018700
SVM	92.04%	92.90%	0.017415

4.2 Conclusion

The implementation and comparison of various machine learning models for this classification task has revealed several interesting insights:

4.2.1 1. PCA Implementation Impact

- The ANN with PCA slightly outperformed the non-PCA version (92.92% vs 92.66%)
- More significantly, PCA reduced the loss function from 0.50 to 0.39
- The dimensional reduction allowed for a more efficient architecture (252/150 nodes vs 700/400 nodes)
- The maintained performance despite reduced complexity suggests effective feature extraction

4.2.2 2. Model Architecture Optimization

- The implementation of dropout (0.3) and regularization successfully addressed overfitting
- The reduced architecture with PCA achieved comparable results with significantly fewer parameters
- The more compact architecture likely contributed to the reduced loss function

4.2.3 3. Comparative Performance

- Random Forest emerged as the top performer with 92.99% test accuracy and 93.45% CV mean accuracy
- Both ANN implementations (with and without PCA) performed competitively with Random Forest
- SVM showed strong performance (92.04% test accuracy, 92.90% CV mean accuracy)
- Logistic Regression significantly underperformed compared to other models

4.2.4 4. Model Stability

- All advanced models (Random Forest, SVM, and both ANNs) achieved >92% accuracy
- Cross-validation standard deviations were consistently low (~0.018), indicating stable performance
- The similar performance across different architectural approaches suggests robust feature relationships

4.2.5 5. Efficiency Considerations

- PCA implementation provided a more efficient solution without sacrificing performance
- The reduced architecture with PCA suggests potential benefits for computational efficiency
- The lower loss function with PCA indicates better model optimization

4.3 Recommendations

1. For production deployment, both the Random Forest and ANN with PCA would be strong candidates:
   - Random Forest for its slightly higher accuracy and proven stability
   - ANN with PCA for its efficient architecture and lower loss function
2. Future optimization could focus on:

- Fine-tuning the PCA variance retention percentage
- Exploring ensemble methods combining the strengths of multiple models
- Further architecture optimization for the neural networks

The results demonstrate that while traditional methods like Random Forest remain competitive, modern approaches with dimensional reduction and neural networks can achieve comparable results with potentially more efficient architectures.