

untitled3

April 30, 2025

```
[1]: import pandas as pd
```

```
[2]: import numpy as np
```

```
[3]: df = pd.read_csv('subject_information.csv')
```

```
[4]: print(f"Dataset shape: {df.shape}")
```

Dataset shape: (60, 5)

```
[5]: print("First 5 rows:\n", df.head(), "\n")
```

First 5 rows:

	Label	Age	Sex	CKD	CKD_Stage
0	CKD_01	52	M	True	2
1	CKD_02	55	M	True	3
2	CKD_03	48	F	True	5
3	CKD_04	66	M	True	4
4	CKD_05	50	M	True	5

```
[6]: print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60 entries, 0 to 59
Data columns (total 5 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Label       60 non-null    object
1   Age         60 non-null    int64
2   Sex         60 non-null    object
3   CKD         60 non-null    bool
4   CKD_Stage   60 non-null    int64
dtypes: bool(1), int64(2), object(2)
memory usage: 2.1+ KB
None
```

```
[23]: if df.isnull().values.any():
        print("Handling missing values...")
        numeric_cols = df.select_dtypes(include=[np.number]).columns
        # Impute numeric columns with median
        df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].median())
        # Drop any remaining rows with missing values
        df = df.dropna()
        print("Missing values imputed or dropped.")
    else:
        print("No missing values found in the dataset.")
```

No missing values found in the dataset.

```
[24]: df_processed = df.drop('Label', axis=1).copy()
df_processed['Sex'] = df_processed['Sex'].map({'M': 1, 'F': 0})
df_processed['CKD'] = df_processed['CKD'].astype(int)

# Separate features and target variable
X = df_processed.drop('CKD', axis=1)
y = df_processed['CKD']

print("After encoding, first 5 rows:\n", df_processed.head(), "\n")
```

After encoding, first 5 rows:

	Age	Sex	CKD	CKD_Stage
0	52	1	1	2
1	55	1	1	3
2	48	0	1	5
3	66	1	1	4
4	50	1	1	5

```
[25]: from sklearn.model_selection import train_test_split

# Split into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)
print(f"Training set size: {X_train.shape[0]} samples; Test set size: {X_test.
↪shape[0]} samples.")
```

Training set size: 48 samples; Test set size: 12 samples.

```
[27]: from sklearn.preprocessing import StandardScaler

# Initialize scaler and fit on training set's numeric features
scaler = StandardScaler()
X_train_scaled = X_train.copy()
```

```

X_test_scaled = X_test.copy()

# Only scale Age and CKD_Stage (Sex is binary 0/1 and doesn't need scaling)
X_train_scaled[['Age', 'CKD_Stage']] = scaler.
    ↪fit_transform(X_train_scaled[['Age', 'CKD_Stage']])
X_test_scaled[['Age', 'CKD_Stage']] = scaler.transform(X_test_scaled[['Age',
    ↪'CKD_Stage']])
print("Features scaled (Age and CKD_Stage).")

```

Features scaled (Age and CKD_Stage).

```

[28]: from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

# Define models and hyperparameter grid
log_model = LogisticRegression(solver='liblinear', max_iter=1000)
log_param_grid = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l1', 'l2'],
    'dual': [False] # 'dual' must be False when using L1 with liblinear solver
}
rf_model = RandomForestClassifier(random_state=42)
rf_param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [None, 3, 5],
    'criterion': ['gini', 'entropy']
}
svm_model = SVC()
svm_param_grid = [
    {'kernel': ['linear'], 'C': [0.1, 1, 10]},
    {'kernel': ['rbf'], 'C': [0.1, 1, 10], 'gamma': [0.01, 0.1, 1]}
]

# Initialize GridSearch for each model
log_grid = GridSearchCV(log_model, log_param_grid, cv=5)
rf_grid = GridSearchCV(rf_model, rf_param_grid, cv=5)
svm_grid = GridSearchCV(svm_model, svm_param_grid, cv=5)

# Train (fit) each model on the training set
print("Training Logistic Regression with GridSearchCV...")
log_grid.fit(X_train_scaled, y_train)
print("Best parameters for Logistic Regression:", log_grid.best_params_)

print("\nTraining Random Forest with GridSearchCV...")
rf_grid.fit(X_train_scaled, y_train)

```

```

print("Best parameters for Random Forest:", rf_grid.best_params_)

print("\nTraining SVM with GridSearchCV...")
svm_grid.fit(X_train_scaled, y_train)
print("Best parameters for SVM:", svm_grid.best_params_, "\n")

```

Training Logistic Regression with GridSearchCV...

Best parameters for Logistic Regression: {'C': 10, 'dual': False, 'penalty': 'l2'}

Training Random Forest with GridSearchCV...

Best parameters for Random Forest: {'criterion': 'gini', 'max_depth': None, 'n_estimators': 50}

Training SVM with GridSearchCV...

Best parameters for SVM: {'C': 10, 'kernel': 'linear'}

```

[29]: from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Best models from grid search
best_log = log_grid.best_estimator_
best_rf = rf_grid.best_estimator_
best_svm = svm_grid.best_estimator_

# Predictions on test data
y_pred_log = best_log.predict(X_test_scaled)
y_pred_rf = best_rf.predict(X_test_scaled)
y_pred_svm = best_svm.predict(X_test_scaled)

# Calculate metrics
metrics = {
    'Model': ['Logistic Regression', 'Random Forest', 'Support Vector Machine'],
    'Accuracy': [
        accuracy_score(y_test, y_pred_log),
        accuracy_score(y_test, y_pred_rf),
        accuracy_score(y_test, y_pred_svm)
    ],
    'Precision': [
        precision_score(y_test, y_pred_log),
        precision_score(y_test, y_pred_rf),
        precision_score(y_test, y_pred_svm)
    ],
    'Recall': [
        recall_score(y_test, y_pred_log),
        recall_score(y_test, y_pred_rf),

```

```

        recall_score(y_test, y_pred_svm)
    ],
    'F1-score': [
        f1_score(y_test, y_pred_log),
        f1_score(y_test, y_pred_rf),
        f1_score(y_test, y_pred_svm)
    ]
}
results_all = pd.DataFrame(metrics).set_index('Model')
print("Model Performance with All Features:")
print(results_all.round(3), "\n")

```

Model Performance with All Features:

	Accuracy	Precision	Recall	F1-score
Model				
Logistic Regression	1.0	1.0	1.0	1.0
Random Forest	1.0	1.0	1.0	1.0
Support Vector Machine	1.0	1.0	1.0	1.0

```

[30]: from sklearn.feature_selection import SelectKBest, chi2, RFE

# SelectKBest with chi-squared to select 2 best features
selector = SelectKBest(score_func=chi2, k=2)
selector.fit(X_train, y_train)
kbest_features = X_train.columns[selector.get_support()]
print("SelectKBest selected features (k=2):", list(kbest_features))

```

SelectKBest selected features (k=2): ['Age', 'CKD_Stage']

```

[31]: rfe_estimator = LogisticRegression(solver='liblinear', penalty='l2', C=1)
rfe = RFE(estimator=rfe_estimator, n_features_to_select=2)
rfe.fit(X_train_scaled, y_train)
rfe_features = X_train.columns[rfe.get_support()]
print("RFE selected features (n=2):", list(rfe_features), "\n")

```

RFE selected features (n=2): ['Age', 'CKD_Stage']

```

[32]: X_train_kbest = X_train_scaled[kbest_features]
X_test_kbest = X_test_scaled[kbest_features]

# Retrain models with selected features
log_grid_k = GridSearchCV(log_model, log_param_grid, cv=5)
rf_grid_k = GridSearchCV(rf_model, rf_param_grid, cv=5)
svm_grid_k = GridSearchCV(svm_model, svm_param_grid, cv=5)

```

```

log_grid_k.fit(X_train_kbest, y_train)
rf_grid_k.fit(X_train_kbest, y_train)
svm_grid_k.fit(X_train_kbest, y_train)

# Best models
best_log_k = log_grid_k.best_estimator_
best_rf_k = rf_grid_k.best_estimator_
best_svm_k = svm_grid_k.best_estimator_

# Test set predictions and metrics
y_pred_log_k = best_log_k.predict(X_test_kbest)
y_pred_rf_k = best_rf_k.predict(X_test_kbest)
y_pred_svm_k = best_svm_k.predict(X_test_kbest)

results_kbest = pd.DataFrame({
    'Model': ['Logistic Regression', 'Random Forest', 'Support Vector Machine'],
    'Accuracy': [
        accuracy_score(y_test, y_pred_log_k),
        accuracy_score(y_test, y_pred_rf_k),
        accuracy_score(y_test, y_pred_svm_k)
    ],
    'Precision': [
        precision_score(y_test, y_pred_log_k),
        precision_score(y_test, y_pred_rf_k),
        precision_score(y_test, y_pred_svm_k)
    ],
    'Recall': [
        recall_score(y_test, y_pred_log_k),
        recall_score(y_test, y_pred_rf_k),
        recall_score(y_test, y_pred_svm_k)
    ],
    'F1-score': [
        f1_score(y_test, y_pred_log_k),
        f1_score(y_test, y_pred_rf_k),
        f1_score(y_test, y_pred_svm_k)
    ]
}).set_index('Model')
print("Model Performance with SelectKBest Features:")
print(results_kbest.round(3), "\n")

```

Model Performance with SelectKBest Features:

	Accuracy	Precision	Recall	F1-score
Model				
Logistic Regression	1.0	1.0	1.0	1.0
Random Forest	1.0	1.0	1.0	1.0
Support Vector Machine	1.0	1.0	1.0	1.0

```

[33]: X_train_rfe = X_train_scaled[rfe_features]
      X_test_rfe = X_test_scaled[rfe_features]

      # Retrain models with RFE selected features
      log_grid_r = GridSearchCV(log_model, log_param_grid, cv=5)
      rf_grid_r = GridSearchCV(rf_model, rf_param_grid, cv=5)
      svm_grid_r = GridSearchCV(svm_model, svm_param_grid, cv=5)

      log_grid_r.fit(X_train_rfe, y_train)
      rf_grid_r.fit(X_train_rfe, y_train)
      svm_grid_r.fit(X_train_rfe, y_train)

      best_log_r = log_grid_r.best_estimator_
      best_rf_r = rf_grid_r.best_estimator_
      best_svm_r = svm_grid_r.best_estimator_

      y_pred_log_r = best_log_r.predict(X_test_rfe)
      y_pred_rf_r = best_rf_r.predict(X_test_rfe)
      y_pred_svm_r = best_svm_r.predict(X_test_rfe)

      results_rfe = pd.DataFrame({
          'Model': ['Logistic Regression', 'Random Forest', 'Support Vector Machine'],
          'Accuracy': [
              accuracy_score(y_test, y_pred_log_r),
              accuracy_score(y_test, y_pred_rf_r),
              accuracy_score(y_test, y_pred_svm_r)
          ],
          'Precision': [
              precision_score(y_test, y_pred_log_r),
              precision_score(y_test, y_pred_rf_r),
              precision_score(y_test, y_pred_svm_r)
          ],
          'Recall': [
              recall_score(y_test, y_pred_log_r),
              recall_score(y_test, y_pred_rf_r),
              recall_score(y_test, y_pred_svm_r)
          ],
          'F1-score': [
              f1_score(y_test, y_pred_log_r),
              f1_score(y_test, y_pred_rf_r),
              f1_score(y_test, y_pred_svm_r)
          ]
      }).set_index('Model')
      print("Model Performance with RFE Features:")
      print(results_rfe.round(3), "\n")

```

Model Performance with RFE Features:

	Accuracy	Precision	Recall	F1-score
Model				
Logistic Regression	1.0	1.0	1.0	1.0
Random Forest	1.0	1.0	1.0	1.0
Support Vector Machine	1.0	1.0	1.0	1.0

```
[34]: df_processed.to_csv('subject_information_processed.csv', index=False)
      print("Processed dataset saved as 'subject_information_processed.csv'.")
```

Processed dataset saved as 'subject_information_processed.csv'.

```
[ ]:
```