Task 1 P, C: ML Development Using Python

Introduction

In this case study we will be using python knowledge with ML libraries to determine and analyze dataset. Dataset that will be used here is of "***Chronic Kidney Disease***". 400 records have been provided in this dataset.

This dataset contains 25 columns, i.e.,

- **Age**: This column contains numerical data and units is in years with missing values. ColumnName: 'age'.
- **Blood Pressure**: This column contains numerical data and unit is in mm/Hg with missing values. ColumnName: 'bp'.
- **Specific Gravity**: This column contains categorical data with unique values as: (1.005,1.010,1.015,1.020,1.025) and some missing values. ColumnName: 'sg'.
- **Albumin**: This column contains categorical data with unique values as: (0,1,2,3,4,5) and some missing values. ColumnName: 'al'.
- **Sugar**: This column contains categorical data with unique values as: (0,1,2,3,4,5) and some missing values. ColumnName: 'su'.
- **Red Blood Cells**: This column contains categorical data with unique values as: (normal, abnormal) and some missing values. ColumnName: 'rbc'.
- **Pus Cell**: This column contains categorical data with unique values as: (normal, abnormal) and some missing values. ColumnName: 'pc'.
- **Pus Cell clumps**: This column contains categorical data with unique values as: (present, notpresent) and some missing values. ColumnName: 'pcc'.
- **Bacteria**: This column contains categorical data with unique values as: (present, notpresent) and some missing values. ColumnName: 'ba'.
- **Blood Glucose Random**: This column contains numerical data and units is in mgs/dl with missing values. ColumnName: 'bgr'.
- **Blood Urea**: This column contains numerical data and units is in mgs/dl with missing values. ColumnName: 'bu'.
- **Serum Creatinine**: This column contains numerical data and units is in mgs/dl with missing values. ColumnName: 'sc'.
- **Sodium**: This column contains numerical data and units is in mEq/L with missing values. ColumnName: 'sod'.
- **Potassium**: This column contains numerical data and units is in mEq/L with missing values. ColumnName: 'pot'.
- **Hemoglobin**: This column contains numerical data and units is in gms with missing values. ColumnName: 'hemo'.
- **Packed Cell Volume**: This column contains numerical data with missing values. ColumnName: 'pcv'.
- **White Blood Cell Count**: This column contains numerical data and units is in cells/cmm with missing values. ColumnName: 'wbcc'.
- **Red Blood Cell Count**: This column contains numerical data and units is in millions/cmm with missing values. ColumnName: 'rbcc'.
- **Hypertension**: This column contains categorical data with unique values as: (yes, no) and some missing values. ColumnName: 'htn'.
- **Diabetes Mellitus**: This column contains categorical data with unique values as: (yes, no) and some missing values. ColumnName: 'dm'.
- **Coronary Artery Disease**: This column contains categorical data with unique values as: (yes, no) and some missing values. ColumnName: 'cad'.
- **Appetite**: This column contains categorical data with unique values as: (good, poor) and some missing values. ColumnName: 'appet'.
- **Pedal Edema**: This column contains categorical data with unique values as: (yes, no) and some missing values. ColumnName: 'pe'.
- **Anemia**: This column contains categorical data with unique values as: (yes, no) and some missing values. ColumnName: 'ane'.
- **Class**: This column contains categorical data with unique values as: (ckd, notckd) and some missing values. ColumnName: 'class'.

Brief information about dataset:

- Dataset contains missing values which has been denoted as '?'.
- Dataset have 24 features and 1 target class. These categories have been divided into as 11 numerical and 14 nominal.

Data Loading and Data Pre-processing

- Extracting data into pandas Dataframe.

  Command:

```
kidney_disease_df = pd.read_csv("chronic_kidney_disease_full.csv", usecols=range(1, 26))
kidney_disease_df.head()
```

  Showing top 5 rows of dataframe:

	'age'	'bp'	'sg'	'al'	'su'	'rbc'	'pc'	'pcc'	'ba'	'bgr'	...	'pcv'	'wbcc'	'rbcc'	'htn'	'dm'	'cad'	'appet'	'pe'	'ane'	'class'
0	48	80	1.020	1	0	?	normal	notpresent	notpresent	121	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	7	50	1.020	4	0	?	normal	notpresent	notpresent	?	...	38	6000	?	no	no	no	good	no	no	ckd
2	62	80	1.010	2	3	normal	normal	notpresent	notpresent	423	...	31	7500	?	no	yes	no	poor	no	yes	ckd
3	48	70	1.005	4	0	normal	abnormal	present	notpresent	117	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	51	80	1.010	2	0	normal	normal	notpresent	notpresent	106	...	35	7300	4.6	no	no	no	good	no	no	ckd

- Rectifying column names, as col names contains quotes, which needs to be removed.

  Command:

```
kidney_disease_df.columns = kidney_disease_df.columns.str.replace("'", "")
kidney_disease_df.head()
```

  Showing top 5 rows of dataframe:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	48	80	1.020	1	0	?	normal	notpresent	notpresent	121	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	7	50	1.020	4	0	?	normal	notpresent	notpresent	?	...	38	6000	?	no	no	no	good	no	no	ckd
2	62	80	1.010	2	3	normal	normal	notpresent	notpresent	423	...	31	7500	?	no	yes	no	poor	no	yes	ckd
3	48	70	1.005	4	0	normal	abnormal	present	notpresent	117	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	51	80	1.010	2	0	normal	normal	notpresent	notpresent	106	...	35	7300	4.6	no	no	no	good	no	no	ckd

- Checking for duplicate rows, but none is present.

  Command:

```
# Checking duplicates in dataframe.
kidney_disease_df[kidney_disease_df.duplicated()]
```

- Replacing missing values i.e. '?' with NAN.

  Command:

```
# Replacing missing values i.e. '?' with NAN
kidney_disease_df = kidney_disease_df.replace("?", np.NaN)

kidney_disease_df.head()
```

  Showing top 5 rows of dataframe:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	48	80	1.020	1	0	NaN	normal	notpresent	notpresent	121	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	7	50	1.020	4	0	NaN	normal	notpresent	notpresent	NaN	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	62	80	1.010	2	3	normal	normal	notpresent	notpresent	423	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	48	70	1.005	4	0	normal	abnormal	present	notpresent	117	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	51	80	1.010	2	0	normal	normal	notpresent	notpresent	106	...	35	7300	4.6	no	no	no	good	no	no	ckd

- As we have 14 categorical data that needs to be encoded, so for this we will be using ‘LabelEncoder’ technique from ‘sklearn’ package. This is a technique that converts categorical variables into numerical values.
  Command:

```
def encode(data):
    """function to encode non-null data"""

    data_no_null = np.array(data.dropna())  # retains only non-null values

    encoded_data = label_encoder.fit_transform(data_no_null)  # encode date

    data.loc[data.notnull()] = np.squeeze(
        encoded_data
    )  # Assign back encoded values to non-null values

    return data


kidney_disease_df[categorical_cols] = kidney_disease_df[categorical_cols].apply(encode)

kidney_disease_df[categorical_cols] = kidney_disease_df[categorical_cols].astype(
    "category"
)

kidney_disease_df.head()
```

  Showing top 5 rows of dataframe:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	48	80	3	1	0	NaN	1	0	0	121	...	44	7800	5.2	1	1	0	0	0	0	0
1	7	50	3	4	0	NaN	1	0	0	NaN	...	38	6000	NaN	0	0	0	0	0	0	0
2	62	80	1	2	3	1	1	0	0	423	...	31	7500	NaN	0	1	0	1	0	1	0
3	48	70	0	4	0	1	0	1	0	117	...	32	6700	3.9	1	0	0	1	1	1	0
4	51	80	1	2	0	1	1	0	0	106	...	35	7300	4.6	0	0	0	0	0	0	0

- As we have ample missing data in every column, so for that we will be using ‘KNNImputer’ technique from ‘sklearn’ package. This is a scikit-learn class that uses the K-Nearest Neighbors (KNN) algorithm to predict or fill in missing values in a dataset. It's a multivariate technique that considers multiple features in the dataset to estimate the missing values.
  Command:

```
def impute(data, col):
    """function to impute null data"""

    result = knn_imputer.fit_transform(data)

    if col in categorical_cols:
        return result.astype(int)

    return np.round(result, 2)


for col in kidney_disease_df.columns:
    kidney_disease_df[[col]] = impute(kidney_disease_df[[col]], col)

kidney_disease_df.head()
```

Showing top 5 rows of dataframe:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	48.0	80.0	3	1	0	0	1	0	0	121.00	...	44.0	7800.0	5.20	1	1	0	0	0	0	0
1	7.0	50.0	3	4	0	0	1	0	0	148.04	...	38.0	6000.0	4.71	0	0	0	0	0	0	0
2	62.0	80.0	1	2	3	1	1	0	0	423.00	...	31.0	7500.0	4.71	0	1	0	1	0	1	0
3	48.0	70.0	0	4	0	1	0	1	0	117.00	...	32.0	6700.0	3.90	1	0	0	1	1	1	0
4	51.0	80.0	1	2	0	1	1	0	0	106.00	...	35.0	7300.0	4.60	0	0	0	0	0	0	0

- As we have 11 non categorical data whose values ranges a lot, so to normalize we will be using ‘StandardScaler’ technique from ‘sklearn’ package. It normalizes features by removing the mean and scaling them to unit variance.

Command:

```
# Fit and transform the scaler to the training data
kidney_disease_df[non_categorical_cols] = scaler.fit_transform(
    kidney_disease_df[non_categorical_cols]
)

kidney_disease_df.head()
```

Showing top 5 rows of dataframe:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	-0.205459	0.262336	3	1	0	0	1	0	-0.241249	-0.361993	...	0.628470	-0.240518	0.585900	1	1	0	0	0	0	0
1	-2.623805	-1.966582	3	4	0	0	1	0	-0.241249	0.000042	...	-0.108551	-0.954786	0.002055	0	0	0	0	0	0	0
2	0.620318	0.262336	1	2	3	1	1	0	-0.241249	3.681436	...	-0.968408	-0.359563	0.002055	0	1	0	1	0	1	0
3	-0.205459	-0.480637	0	4	0	1	0	1	-0.241249	-0.415548	...	-0.845571	-0.677015	-0.963076	1	0	0	1	1	1	0
4	-0.028507	0.262336	1	2	0	1	1	0	-0.241249	-0.562825	...	-0.477061	-0.438926	-0.129012	0	0	0	0	0	0	0

Data Visualization

- Categorical Pie chart visualize

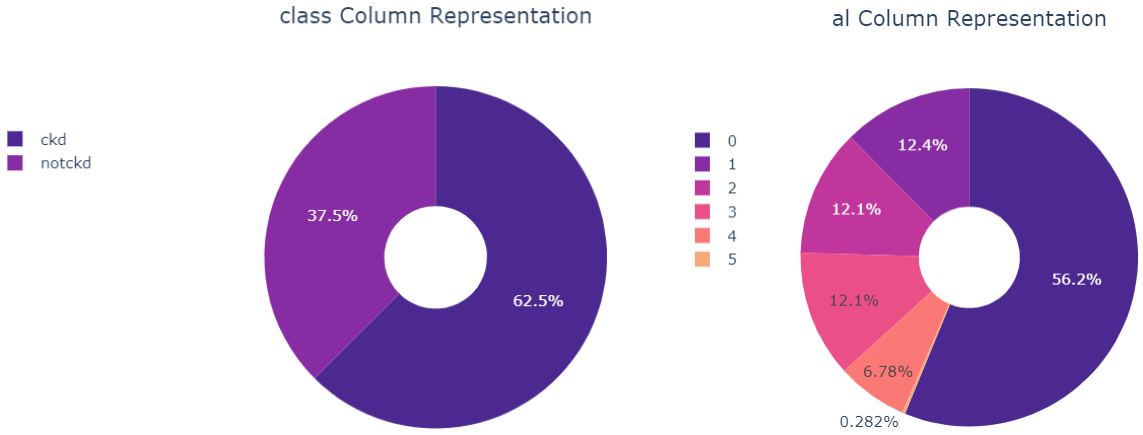
Command:

```
# Pie Chart Visualization
for pie_col_name in categorical_cols:
    pie_graph = px.pie(
        kidney_disease_df,
        hole=0.3,
        values=kidney_disease_df[pie_col_name].value_counts(),
        names=kidney_disease_df[pie_col_name].value_counts().index,
        color_discrete_sequence=px.colors.sequential.turbid_r,
        title=pie_col_name + " Column Representation",
    )

    # Update the layout to center the title
    pie_graph.update_layout(
        title_x=0.5, legend=dict(yanchor="top", y=0.9, xanchor="center", x=0.25)
    )

    pie_graph.show()
```

Visualizing sample pie charts:



- Heatmap visualization

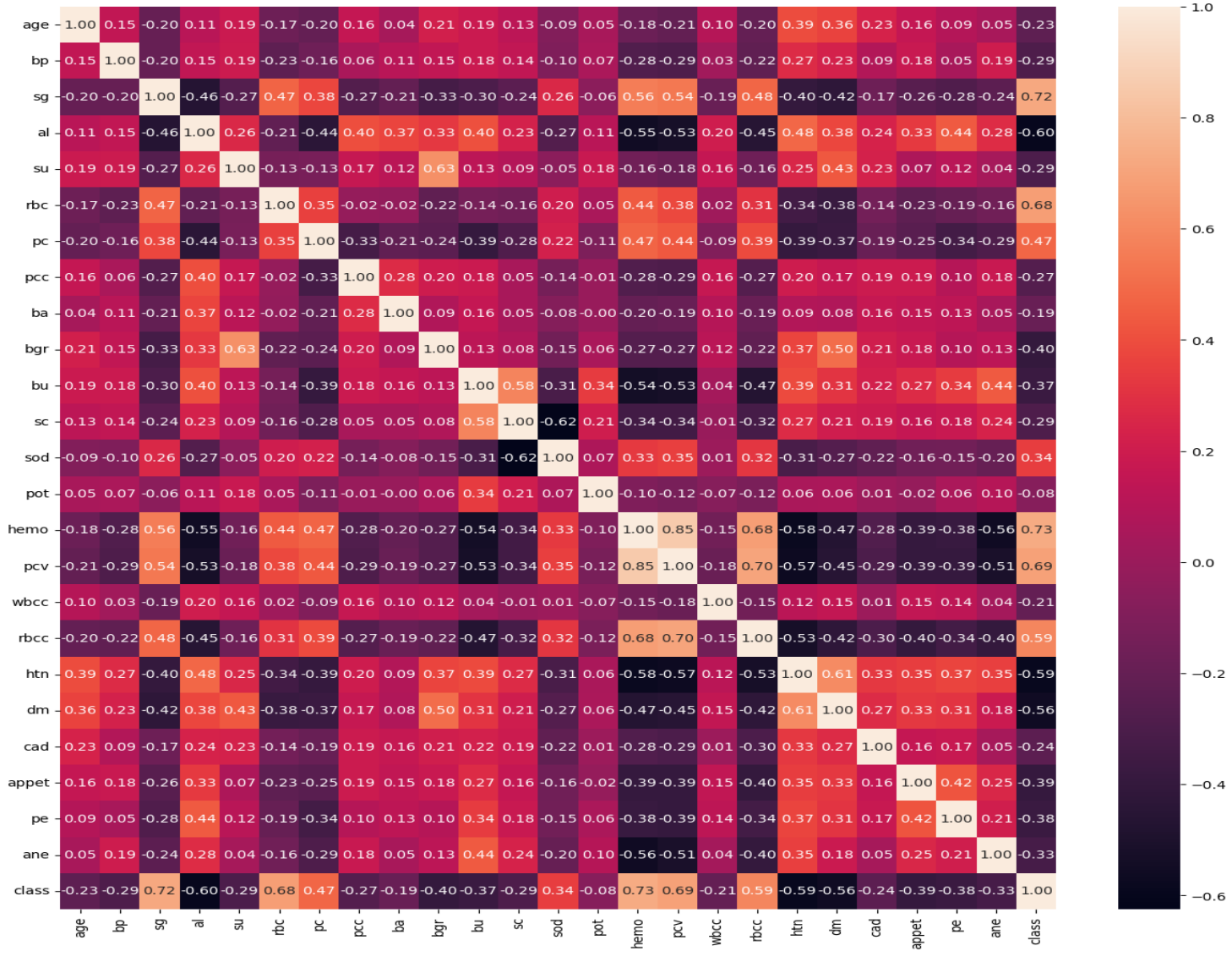
Command:

```
# Correlation Matrix
plt.figure(figsize=(15, 15))
sns.heatmap(kidney_disease_df.corr(), annot=True, fmt=".2f")

plt.xticks(rotation="90")
plt.yticks(rotation="0")

plt.show()
```

Visualizing heatmap:



Data Split

Now we will be splitting our dataframe into 80% training dataset and 20% as testing dataset. To achieve this, we will be using `train_test_split' from 'sklearn' package. This is a function in the scikit-learn library that splits a dataset into two sets: a training set and a test set. The training set is used to fit a machine learning model, while the test set is used to evaluate the model's performance.

Command:

```
X_train, X_test, y_train, y_test = train_test_split(
    kidney_disease_df.drop(columns="class", axis=1),
    kidney_disease_df["class"],
    test_size=0.2,
    random_state=42,
)

X_train.shape, X_test.shape
```

Data Modelling

In this we will be using 2 ML models based on Decision tree and Random Forest.

- DecisionTreeClassifier: It is a machine learning algorithm that predicts the probability of a categorical dependent variable. It's a set of if/else or Yes/No questions that can perform multi-class classification on a dataset.
- RandomForestClassifier: It is a supervised machine learning algorithm that uses decision trees to classify, regress, and perform other tasks. It's a type of ensemble-based learning method that uses a random vector sampled from the input vector to generate each decision tree.

To achieve the best model, we have used 'GridSearchCV' technique from 'sklearn' package. It is a hyperparameter tuning technique used in machine learning to find the best combination of hyperparameters for a given model. Hyperparameters are variables that are not learned by the model, but rather set by the user before training.

Defining model and param details JSON for our 'GridSearchCV'.

Command:

```
model_details = {
    "DecisionTree_Classifier": DecisionTreeClassifier(random_state=42),
    "RandomForest_Classifier": RandomForestClassifier(random_state=42),
}

param_details = {
    "DecisionTree_Classifier": {
        "ccp_alpha": [0.1, 0.01, 0.001],
        "max_depth": list(range(0, 9)),
        "criterion": ["gini", "entropy"],
    },
    "RandomForest_Classifier": {
        "n_estimators": list(range(10, 50, 5)),
        "max_depth": list(range(0, 9)),
        "criterion": ["gini", "entropy"],
    },
}
```

Here we will be defining our ` GridSearchCV', which will be providing best model that can be used. Once best estimator is extracted, will be displaying performance of our model.

We have used different performance measures like, accuracy, ROC, precision, recall and f1 score.

Command:

```
print("Running GridSearchCV for %s." % key)

grid_search = GridSearchCV(
    model_details.get(key), param_details.get(key), cv=10, n_jobs=-1, refit=True
)
grid_search.fit(X_train, y_train)
print(f"GridSearchCV best params for {key} are {grid_search.best_params_}")

predicted_model = grid_search.best_estimator_

predicted_model.fit(X_train, y_train)

y_pred = predicted_model.predict(X_test)
y_train_pred = predicted_model.predict(X_train)

train_model_lists.append(
    [
        key,
        accuracy_score(y_train, y_train_pred),
        accuracy_score(y_test, y_pred),
        roc_auc_score(y_test, y_pred),
        precision_score(y_test, y_pred),
        recall_score(y_test, y_pred),
        f1_score(y_test, y_pred),
        predicted_model,
    ]
)
print(f"Determined model: {predicted_model}")

plt.figure(figsize=(4, 4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt=".2f")
plt.show()

print("\nClassification Matrix:\n", classification_report(y_test, y_pred))

print("GridSearchCV for %s completed.\n" % key)
```

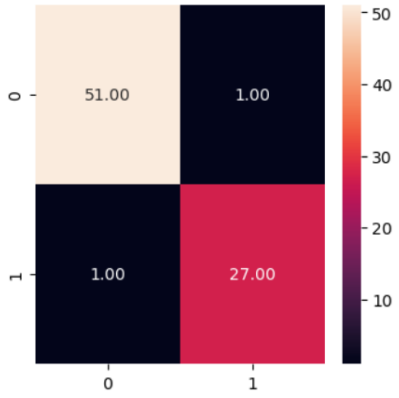
Printing output for Decision tree model:

```
Running GridSearchCV for DecisionTree_Classifier.
GridSearchCV best params for DecisionTree_Classifier are {'ccp_alpha': 0.001, 'criterion': 'entropy', 'max_depth': 6}
Determined model: DecisionTreeClassifier(ccp_alpha=0.001, criterion='entropy', max_depth=6,
                       random_state=42)
```

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

           0       0.98      0.98      0.98        52
           1       0.96      0.96      0.96        28

    accuracy                           0.97        80
   macro avg       0.97      0.97      0.97        80
weighted avg       0.97      0.97      0.97        80
```



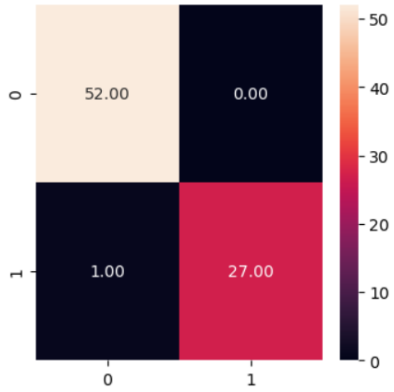
Printing output for Random search tree model:

```
Running GridSearchCV for RandomForest_Classifier.
GridSearchCV best params for RandomForest_Classifier are {'criterion': 'gini', 'max_depth': 7, 'n_estimators': 20}
Determined model: RandomForestClassifier(max_depth=7, n_estimators=20, random_state=42)
```

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

           0       0.98      1.00      0.99        52
           1       1.00      0.96      0.98        28

    accuracy                           0.99        80
   macro avg       0.99      0.98      0.99        80
weighted avg       0.99      0.99      0.99        80
```



Sorted models with performance ("Recall", "F1 Score").

	Model_Name	Train_Accuracy	Test_Accuracy	ROC_AUC	Precision	Recall	F1 Score
1	RandomForest_Classifier	1.0	0.9875	0.982143	1.000000	0.964286	0.981818
0	DecisionTree_Classifier	1.0	0.9750	0.972527	0.964286	0.964286	0.964286

As per the above table, we can conclude that RandomForest_Classifier has provided best performance and can be used for our further predictions or analysis.

Model Save & Load

Pickling is the process of converting a Python object hierarchy into a byte stream, and "unpickling" is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy.

- Saving model

```
# create an iterator object with write permission - model.pkl
with open("model_pkl", "wb") as files:
    pickle.dump(best_model, files)
```

- Loading model

```
with open("model_pkl", "rb") as f:
    pickle_model = pickle.load(f)
print(pickle_model)
```

✓ 0.0s

```
RandomForestClassifier(max_depth=7, n_estimators=20, random_state=42)
```

Limitation

- In this 'Chronic Kidney Disease' dataset have very less records, because of which our model won't be much helpful.
- We can use SMOTE technique also, to increase the dataset records.
- Dataset have multi null values, for which imputation has been done, but that's not the real/original values.

Conclusion

In this case study we have compared 2 ML models, from which 'RandomForestClassifier' has provided best performance.

For our modelling we can use different ML models also, that can provide much better prediction performance, like Support Vector Machine, K-Nearest Neighbours, Stochastic Gradient Descent, and more.

Grade

Here am targeting for 'Pass' and 'Credit' grade.

References

- https://olympus.mygreatlearning.com/dashboard
- https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
- https://www.geeksforgeeks.org/
- https://www.google.com/