

Indian Institute of Technology Jodhpur
MLOps

Assignment 2
ML Ops

Submitted by: Jojo Joseph (G23AI2100)

Google Colab :

https://colab.research.google.com/drive/12_uCB1524aOU9-Bp2GtbIeoMkBkHAqL?usp=sharing

gitHub : <https://github.com/IITJPGD/MLOpsAssignment2>

MLOps Assignment

Q1 Use the attached dataset which is a classification problem with Indianliver pateints whether liver patent or not as a output variable

Ans

```
import pandas as pd
url = "https://raw.githubusercontent.com/IITJPGD/MLOpsAssignment2/main/indian_liver_patientmissing.csv"
df = pd.read_csv(url)
print(df.head())
print(df.info())
print(df.describe())
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	LiverPateientOrNot
0	65	Female	0.7	0.1	187	16	18	6.8	3.3	0.90	0
1	62	Male	10.9	5.5	699	64	100	7.5	3.2	0.74	1
2	62	Male	7.3	4.1	490	60	68	7	3.3	0.89	1
3	58	Male	1	0.4	182	14	20	6.8	3.4	1.00	1
4	72	Male	3.9	2	195	27	59	7.3	2.4	0.40	1

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
 # Column Non-Null Count Dtype

The screenshot shows a Jupyter Notebook interface with the title "mllops_assignment2_g23ai2100.ipynb". The notebook has a dark theme and displays the following code and output:

```
+ Code + Text
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
 0   Age              583 non-null    int64  
 1   Gender            583 non-null    object 
 2   Total_Bilirubin  583 non-null    object 
 3   Direct_Bilirubin 583 non-null    object 
 4   Alkaline_Phosphotase 583 non-null    object 
 5   Alamine_Aminotransferase 583 non-null    object 
 6   Aspartate_Aminotransferase 583 non-null    object 
 7   Total_Protiens   583 non-null    object 
 8   Albumin           583 non-null    float64 
 9   Albumin_and_Globulin_Ratio 579 non-null    float64 
 10  LiverPateientOrNot 1meansNO  583 non-null    int64  
dtypes: float64(2), int64(2), object(7)
memory usage: 50.2+ KB
None
Age      Age      Albumin  Albumin_and_Globulin_Ratio \
count  583.000000  583.000000  579.000000
mean   44.746141  3.141852   0.947064
std    16.189833  0.795519   0.319592
min    4.000000   0.900000   0.300000
25%   33.000000   2.600000   0.700000
50%   45.000000   3.100000   0.930000
75%   58.000000   3.800000   1.100000
max   90.000000   5.500000   2.800000

LiverPateientOrNot 1meansNO
count  583.000000
mean   1.286449
std    0.452490
min    1.000000
25%   1.000000
50%   1.000000
75%   2.000000
max   2.000000
```

The notebook indicates "0s completed at 11:34".

Q2 Do descriptive analytics of all columns by first finding discrete and continuous columns

Ans

mlops_assignment2_g23ai2113.ipynb

	580	52	Male	0.8	0.2	245	48	49	6.4	3.2
[]	581	31	Male	1.3	0.5	184	29	32	6.8	3.4
{x}	582	38	Male	1	0.3	216	21	24	7.3	4.4

583 rows x 11 columns

```
# Function to identify discrete and continuous variables
def identify_variable_types_g23ai2113(data, threshold=10):
    discrete_vars_g23ai2113 = []
    continuous_vars_g23ai2113 = []

    for col in data.columns:
        unique_values = data[col].nunique()
        if unique_values <= threshold:
            discrete_vars_g23ai2113.append(col)
        else:
            continuous_vars_g23ai2113.append(col)

    return discrete_vars_g23ai2113, continuous_vars_g23ai2113

# Set a threshold for the number of unique values
threshold_g23ai2113 = 10
discrete_vars_g23ai2113, continuous_vars_g23ai2113 = identify_variable_types_g23ai2113(df_g23ai2113, threshold_g23ai2113)

print("Discrete Variables:", discrete_vars_g23ai2113)
print("Continuous Variables:", continuous_vars_g23ai2113)
```

Discrete Variables: ['Gender', 'LiverPateientOrNot 1meansNO']
 Continuous Variables: ['Age', 'Total_Bilirubin', 'Direct_Bilirubin', 'Alkaline_Phosphotase', 'Alamine_Aminotransferase', 'Aspartate_Aminotransferase', 'Total_Protiens']

Q3 Do imputation of relevant missing values

Ans

```
df['Gender'] = df['Gender'].replace(['', None], 'Male') # Assuming default is 'Male' # Step 1: Replace empty or missing value

gender_map_g23ai2113 = {'Male': 1, 'Female': 0} # Step 2: Map gender to integers
df['Gender'] = df['Gender'].map(gender_map_g23ai2113)

object_columns_g23ai2113 = ['Total_Bilirubin', 'Direct_Bilirubin', 'Alkaline_Phosphotase',
                             'Alamine_Aminotransferase', 'Aspartate_Aminotransferase', 'Total_Protiens']

for col in object_columns_g23ai2113:
    df[col] = pd.to_numeric(df[col], errors='coerce')

# missing values count
print(df.isnull().sum())

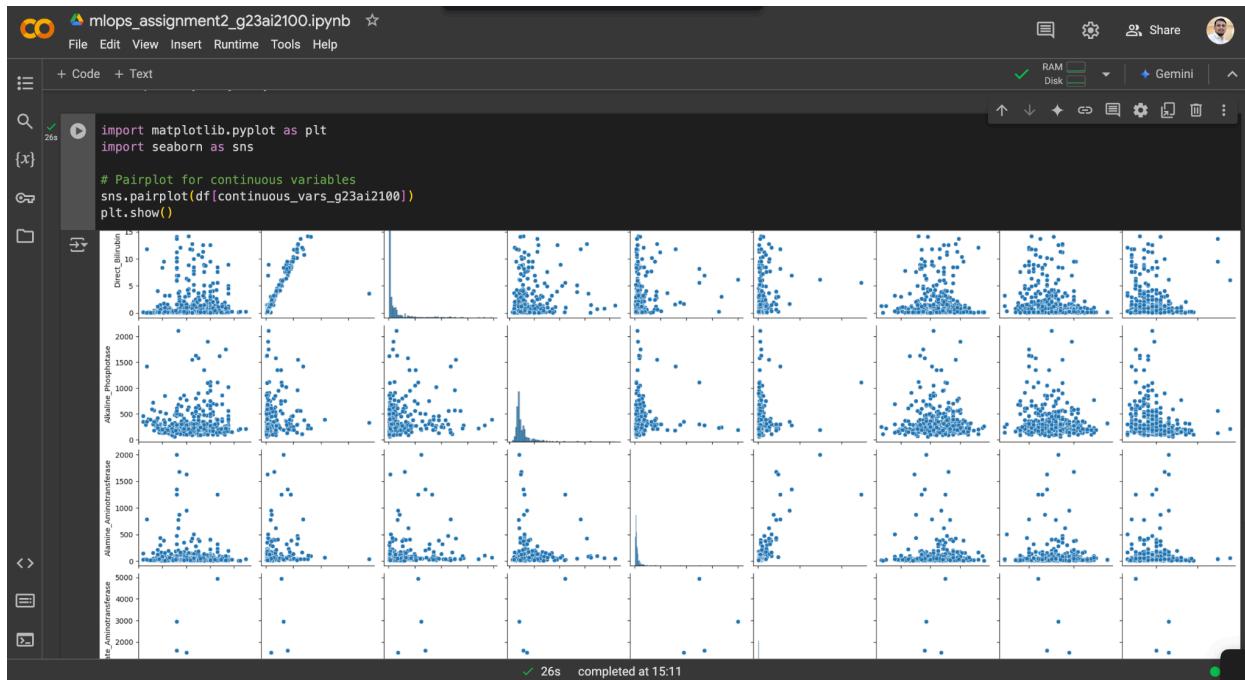
# Impute missing values
df.fillna(df.mean(), inplace=True) # For continuous columns
df.fillna(df.mode().iloc[0], inplace=True) # For discrete columns
```

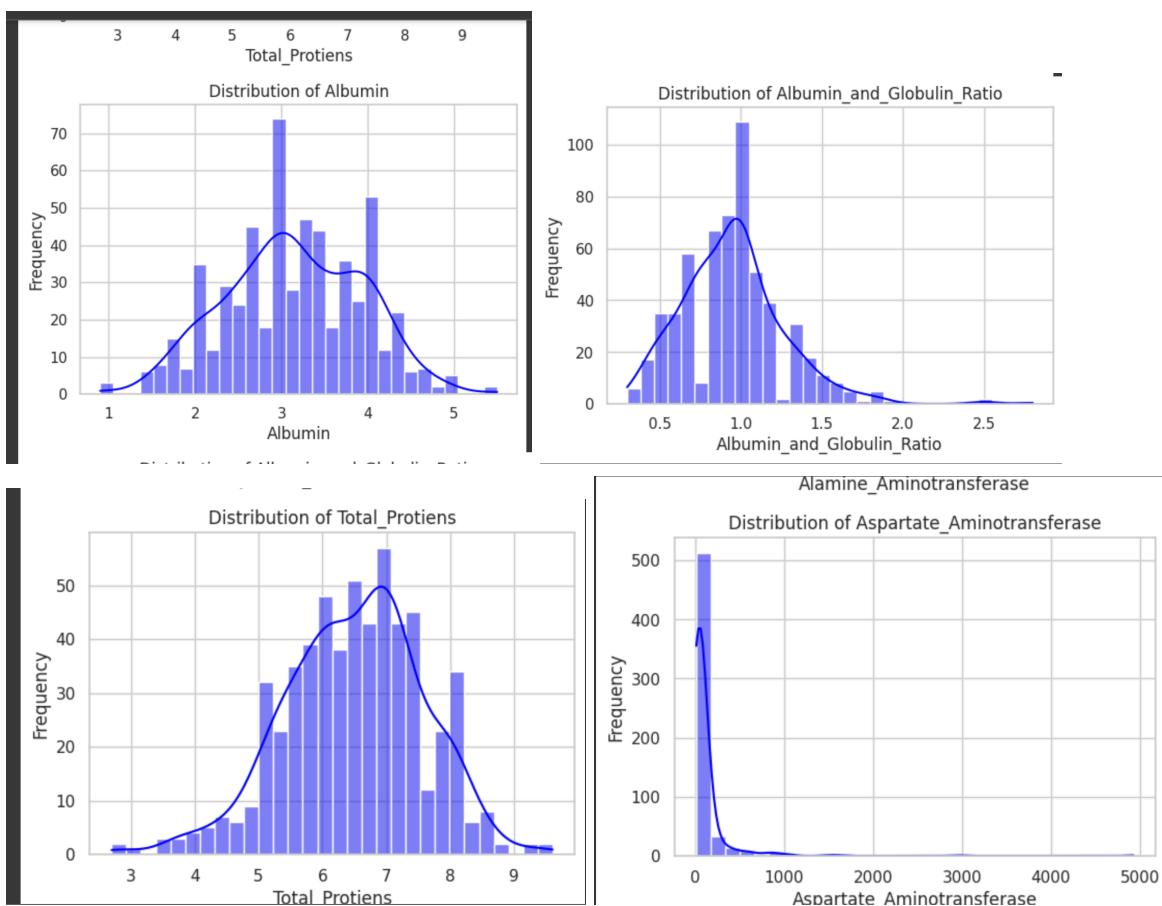
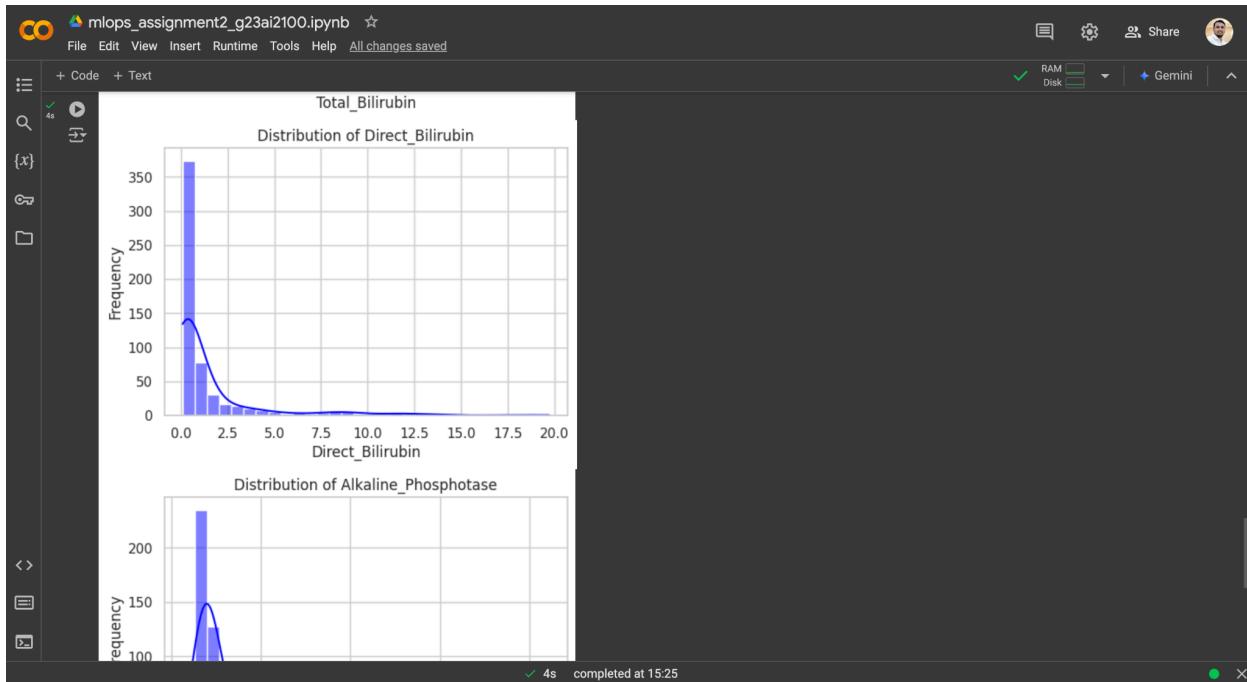
	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	LiverPateientOrNot 1meansNO
	0	583	0	0	0	0	0	0	0	0	0

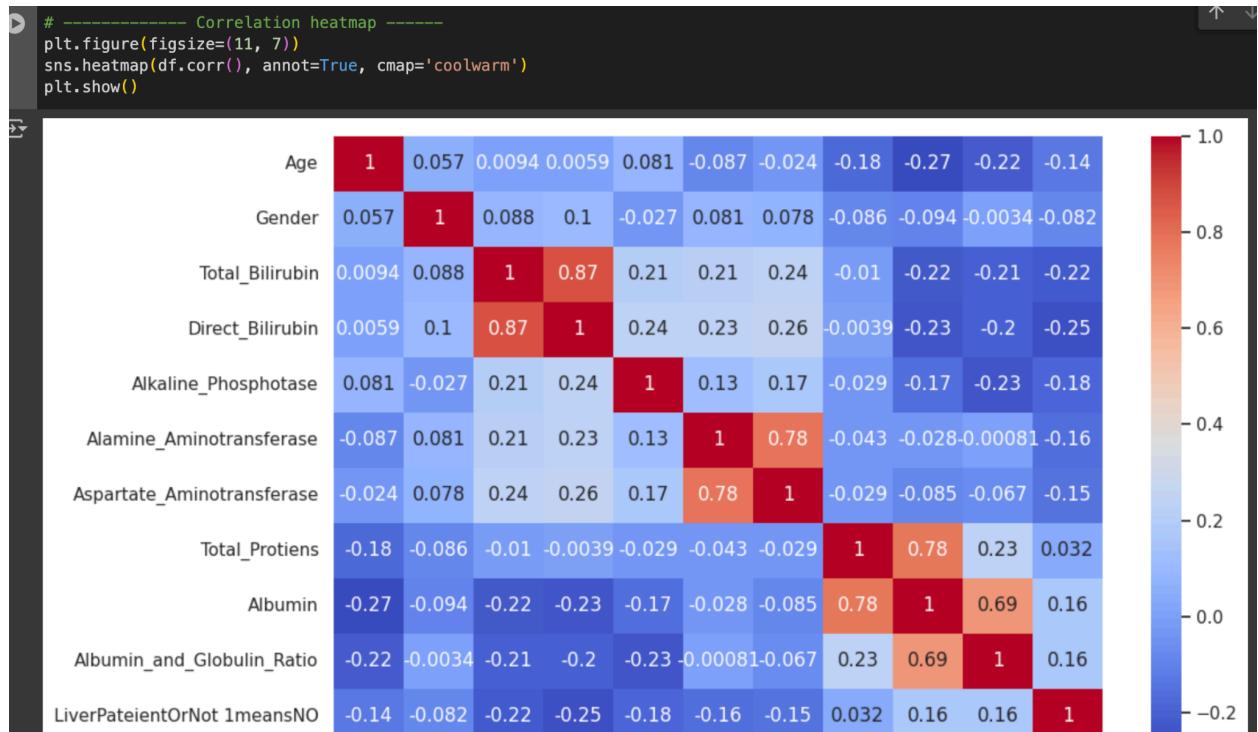
✓ 0s completed at 11:49

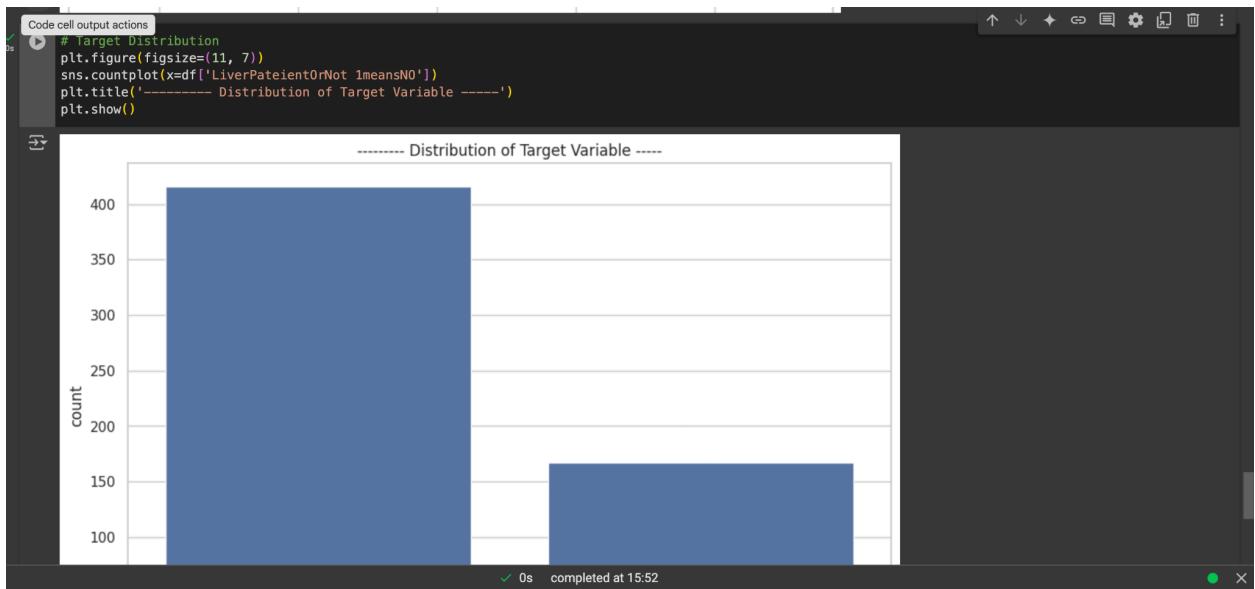
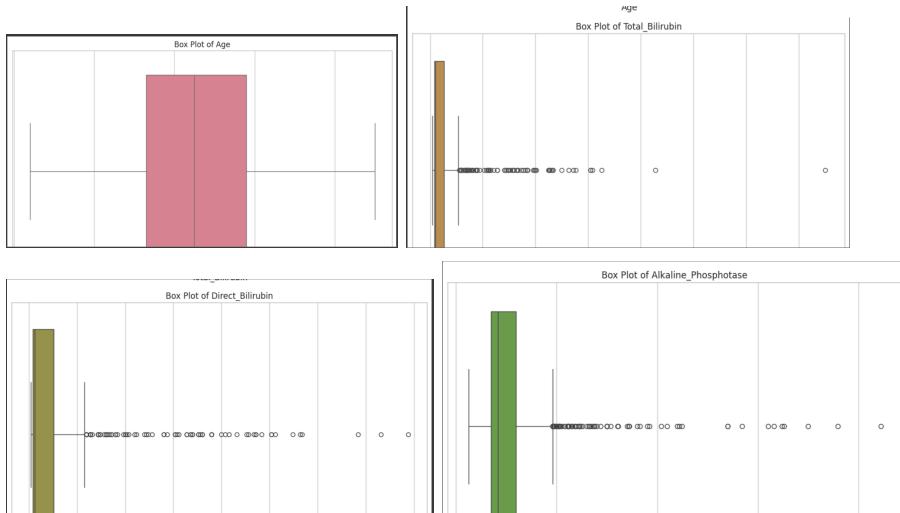
Q4 Do exploratory analysis by different visualizations

Ans









Q5 Perform different feature selection algorithms including pairwise correlation

Ans

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** mlops_assignment2_g23ai2100.ipynb
- File Menu:** File, Edit, View, Insert, Runtime, Tools, Help
- Status Bar:** All changes saved, RAM Disk, Gemini
- Code Cell:** Contains Python code for feature selection and correlation analysis.
- Output Cell:** Displays the results of the executed code, including:
 - Discrete Feature Scores: [0.96451837 92.62933333]
 - Continuous Feature Scores: [11.1714293 29.75267932 37.94378977 20.56733999 15.74457342 13.12464455 0.57976987 15.53743097 15.72213621]
 - A table showing the correlation matrix between features:
- Table Output:** Correlation matrix (partial view)

	Age	Gender	Total_Bilirubin
Age	1.000000	0.056560	0.009429
Gender	0.056560	1.000000	0.088078
Total_Bilirubin	0.009429	0.088078	1.000000
Direct_Bilirubin	0.005871	0.100762	0.873330
Alkaline_Phosphatase	0.080538	-0.027462	0.205442
Alamine_Aminotransferase	-0.087203	0.081393	0.213265
Aspartate_Aminotransferase	-0.023767	0.077748	0.236509
Total_Protiens	-0.177874	-0.086323	-0.010337
Albumin	-0.265924	-0.093799	-0.221874
Albumin_and_Globulin_Ratio	-0.216089	-0.003404	-0.205840

Q6 Use optuna to fine tune decision tree, and random forest classification algorithms to find the best hyperparameter combination

Ans

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** mllops_assignment2_g23ai2100.ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help, All changes saved.
- Code Cell:** Contains Python code for machine learning experiments using Optuna, scikit-learn, and pandas. The code defines objective functions for Decision Tree and Random Forest classifiers, uses train-test split, and performs hyperparameter optimization.
- Runtime Status:** RAM: 1.2 GB, Disk: 1.2 GB, Gemini: 1.2 GB.
- Bottom Status:** ✓ 37s completed at 19:01

OutPut :

```
Best Random Forest Params: { 'n_estimators': 458, 'max_depth': 3,
'min_samples_split': 8, 'min_samples_leaf': 9}
```

Q7 Do different feature engineering steps of PCA and one hot encoding and then do classification and compare without feature engineering and with feature engineering

Ans

The screenshot shows a Jupyter Notebook interface with a dark theme. The code cell contains Python code for feature engineering:

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

onehot_encoder = OneHotEncoder()
pca = PCA(n_components=5) # PCA - continuous columns

continuous_vars = [col for col in continuous_vars_g23ai2100 if col in X_train.columns] # Ensure that continuous_vars and discrete_vars only contain columns
discrete_vars = [col for col in discrete_vars_g23ai2100 if col in X_train.columns]

preprocessor = ColumnTransformer(
    transformers=[
        ('num', pca, continuous_vars),
        ('cat', onehot_encoder, discrete_vars)
    ])

pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('classifier', DecisionTreeClassifier(**study_dt.best_params))]) # Create a pipeline

pipeline.fit(X_train, y_train)

# Predict/ evaluate
y_pred = pipeline.predict(X_test)
print("Accuracy(Feature Engineering):", accuracy_score(y_test, y_pred))
```

The output cell shows the accuracy of the Feature Engineering model:

```
Accuracy(Feature Engineering): 0.6923076923076923
```

Output : Accuracy (Feature Engineering) : 0.6923076923076923

Q8 Use MLFlow to run different runs of the same algorithms Decision Tree , and Random Forest and capture the input hyperparameters and output metrics of the same

Ans

The screenshot shows a Jupyter Notebook interface with a dark theme. The code cell contains Python code for logging models with MLflow:

```
import mlflow.sklearn
mlflow.start_run()
with mlflow.start_run():
    mlflow.log_params(study_dt.best_params)
    mlflow.sklearn.log_model(pipeline, "decision_tree_model")
    mlflow.log_metric("accuracy", accuracy_score(y_test, y_pred))

with mlflow.start_run():
    mlflow.log_params(study_rf.best_params)
    pipeline_rf = Pipeline(steps=[('preprocessor', preprocessor),
                                  ('classifier', RandomForestClassifier(**study_rf.best_params))])
    pipeline_rf.fit(X_train, y_train)
    y_pred_rf = pipeline_rf.predict(X_test)
    mlflow.log_metric("accuracy", accuracy_score(y_test, y_pred_rf))
    mlflow.sklearn.log_model(pipeline_rf, "random_forest_model")
```

MLflow logs:

```
2024/12/15 13:49:46 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the m
2024/12/15 13:49:50 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the m
```

```
import mlflow
from mlflow.tracking import MlflowClient

ml_client = MlflowClient()

experiment_name = "pateints-assignment"
experiment = ml_client.get_experiment_by_name(experiment_name)
experiment_id = experiment.experiment_id

# Fetch all runs for the experiment
runs = ml_client.search_runs(
    experiment_ids=[experiment_id])
```

Output

Run ID: 9fa6362996274af7ab6a9db912cab030

Parameters: {'n_estimators': '137', 'min_samples_leaf': '12', 'max_depth': '19', 'min_samples_split': '7'}

Metrics: {'accuracy': 0.7692307692307693}

Artifacts: [<FileInfo: file_size=None, is_dir=True, path='random_forest_model'>]

Run ID: e9ee433632254247889d5ce41f135a56

Parameters: {'min_samples_leaf': '19', 'max_depth': '23', 'min_samples_split': '2'}

Metrics: {'accuracy': 0.7008547008547008}

Artifacts: [<FileInfo: file_size=None, is_dir=True, path='decision_tree_model'>]