```python
#importing lib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_excel('/content/E Commerce Dataset.xlsx', sheet_name='E
Comm')
df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 5630,\n  \"fields\":
[\n    {\n      \"column\": \"CustomerID\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1625,\n        \"min\":
50001,\n        \"max\": 55630,\n        \"num_unique_values\": 5630,\
n        \"samples\": [\n          54332,\n          51989,\n
53444\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Churn\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          0,\n
1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Tenure\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 8.557240984165002,\n        \"min\": 0.0,\n        \"max\":
61.0,\n        \"num_unique_values\": 36,\n        \"samples\": [\n
61.0,\n          1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PreferredLoginDevice\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"Mobile Phone\",\n          \"Phone\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CityTier\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
\"min\": 1,\n        \"max\": 3,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          3,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"WarehouseToHome\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
8.53147518676263,\n        \"min\": 5.0,\n        \"max\": 127.0,\n
\"num_unique_values\": 34,\n        \"samples\": [\n          14.0,\n
23.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PreferredPaymentMode\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          \"Debit Card\",\n          \"UPI\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Gender\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
2,\n        \"samples\": [\n          \"Male\",\n          \"Female\"\
n        ],\n        \"semantic_type\": \"\",\n
```

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"HourSpendOnApp\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.7219258499760615,\n        \"min\":
0.0,\n        \"max\": 5.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          3.0,\n          2.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"NumberOfDeviceRegistered\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1,\n        \"min\": 1,\n        \"max\": 6,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          3,\n
4\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PreferedOrderCat\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 6,\n        \"samples\":
[\n          \"Laptop & Accessory\",\n          \"Mobile\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"SatisfactionScore\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1,\n        \"min\": 1,\n
\"max\": 5,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          3,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"MaritalStatus\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"Single\",\n          \"Divorced\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"NumberOfAddress\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2,\n        \"min\": 1,\n        \"max\": 22,\n
\"num_unique_values\": 15,\n        \"samples\": [\n          5,\n
21\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Complain\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"OrderAmountHikeFromlastYear\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 3.6754854627464644,\
n        \"min\": 11.0,\n        \"max\": 26.0,\n
\"num_unique_values\": 16,\n        \"samples\": [\n          11.0,\n
15.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"CouponUsed\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1.8946214472186502,\n        \"min\":
0.0,\n        \"max\": 16.0,\n        \"num_unique_values\": 17,\n
\"samples\": [\n          1.0,\n          0.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"OrderCount\",\n
```

```
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2.9396795481512608,\n        \"min\": 1.0,\n        \"max\": 16.0,\n
\"num_unique_values\": 16,\n        \"samples\": [\n          1.0,\n
6.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DaySinceLastOrder\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3.6544331967013357,\n        \"min\":
0.0,\n        \"max\": 46.0,\n        \"num_unique_values\": 22,\n
\"samples\": [\n          5.0,\n          13.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CashbackAmount\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
49.20703617486409,\n        \"min\": 0.0,\n        \"max\": 324.99,\n
\"num_unique_values\": 2586,\n        \"samples\": [\n
125.19,\n          137.32\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```


```
# eda
df.shape
```

```
(5630, 20)
```

```
df.isnull().sum()
```

```
CustomerID                       0
Churn                            0
Tenure                         264
PreferredLoginDevice             0
CityTier                         0
WarehouseToHome                251
PreferredPaymentMode             0
Gender                           0
HourSpendOnApp                 255
NumberOfDeviceRegistered         0
PreferedOrderCat                 0
SatisfactionScore                0
MaritalStatus                    0
NumberOfAddress                  0
Complain                         0
OrderAmountHikeFromlastYear    265
CouponUsed                     256
OrderCount                     258
DaySinceLastOrder              307
CashbackAmount                   0
dtype: int64
```

```
df.dtypes
```

```
CustomerID                       int64
Churn                            int64
```

```
Tenure                              float64
PreferredLoginDevice                 object
CityTier                              int64
WarehouseToHome                     float64
PreferredPaymentMode                 object
Gender                               object
HourSpendOnApp                      float64
NumberOfDeviceRegistered              int64
PreferedOrderCat                     object
SatisfactionScore                     int64
MaritalStatus                        object
NumberOfAddress                       int64
Complain                              int64
OrderAmountHikeFromlastYear         float64
CouponUsed                          float64
OrderCount                          float64
DaySinceLastOrder                   float64
CashbackAmount                      float64
dtype: object
```

```
#treating null values
df.drop(['CustomerID'],axis=1,inplace=True)

for i in df.columns:
    if df[i].isnull().sum() > 0:
        print(i)
        print('the total null values are:', df[i].isnull().sum())
        print('the datatype is', df[i].dtypes)
        print()
```

```
Tenure
the total null values are: 264
the datatype is float64

WarehouseToHome
the total null values are: 251
the datatype is float64

HourSpendOnApp
the total null values are: 255
the datatype is float64

OrderAmountHikeFromlastYear
the total null values are: 265
the datatype is float64

CouponUsed
the total null values are: 256
the datatype is float64
```

```
OrderCount
the total null values are: 258
the datatype is float64

DaySinceLastOrder
the total null values are: 307
the datatype is float64
```

```python
df['Churn'] = df['Churn'].astype('object')
df['CityTier'] = df['CityTier'].astype('object')

df.describe().transpose()
```

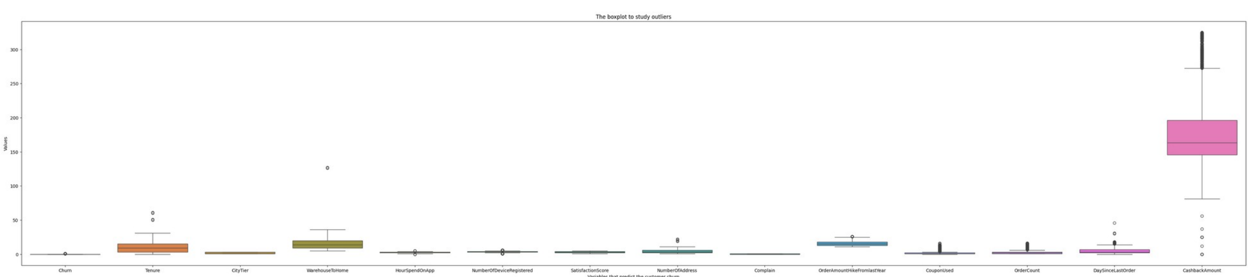
{"summary":"{\n \"name\": \"df\",\n \"rows\": 12,\n \"fields\": [\n {\n   \"column\": \"count\",\n   \"properties\": {\n \"dtype\": \"number\",\n   \"std\": 137.25711330024083,\n \"min\": 5323.0,\n   \"max\": 5630.0,\n \"num_unique_values\": 8,\n   \"samples\": [\n   5379.0,\n 5374.0,\n   5366.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n \"column\": \"mean\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 49.7219396003961,\n   \"min\": 0.2849023090586146,\n   \"max\": 177.22303019538188,\n \"num_unique_values\": 12,\n   \"samples\": [\n 4.543490512868683,\n   3.0080044676098288,\n 10.189899366380917\n   ],\n   \"semantic_type\": \"\",\n \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"std\",\n   \"properties\": {\n   \"dtype\": \"number\",\n \"std\": 13.551471831285205,\n   \"min\": 0.4514079937386503,\n \"max\": 49.20703617486409,\n   \"num_unique_values\": 12,\n \"samples\": [\n   3.6544331967013357,\n 2.9396795481512608,\n   8.557240984165002\n   ],\n \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"min\",\n   \"properties\": {\n \"dtype\": \"number\",\n   \"std\": 3.256694736394648,\n \"min\": 0.0,\n   \"max\": 11.0,\n   \"num_unique_values\": 4,\n   \"samples\": [\n   5.0,\n   11.0,\n 0.0\n   ],\n   \"semantic_type\": \"\",\n \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"25%\",\n   \"properties\": {\n   \"dtype\": \"number\",\n \"std\": 41.281906108726346,\n   \"min\": 0.0,\n   \"max\": 145.77,\n   \"num_unique_values\": 7,\n   \"samples\": [\n 2.0,\n   9.0,\n   1.0\n   ],\n \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"50%\",\n   \"properties\": {\n \"dtype\": \"number\",\n   \"std\": 45.900833688870335,\n \"min\": 0.0,\n   \"max\": 163.28,\n \"num_unique_values\": 9,\n   \"samples\": [\n   2.0,\n 14.0,\n   15.0\n   ],\n   \"semantic_type\": \"\",\n

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"75%\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 54.88695093986711,\n        \"min\": 1.0,\n        \"max\":
196.3925,\n        \"num_unique_values\": 10,\n        \"samples\": [\
n          7.0,\n          20.0,\n          1.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"max\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 92.1871910308125,\n
\"min\": 1.0,\n        \"max\": 324.99,\n
\"num_unique_values\": 10,\n        \"samples\": [\n          46.0,\n
127.0,\n          1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```



```
#treating outliers
for i in df.columns:
    if df[i].isnull().sum() > 0:
        df[i].fillna(df[i].median(),inplace=True)

plt.figure(figsize=(50,10))
sns.boxplot(data=df)
plt.title('The boxplot to study outliers')
plt.xlabel('Variables that predict the customer churn')
plt.ylabel('Values')
```

```
Text(0, 0.5, 'Values')
```



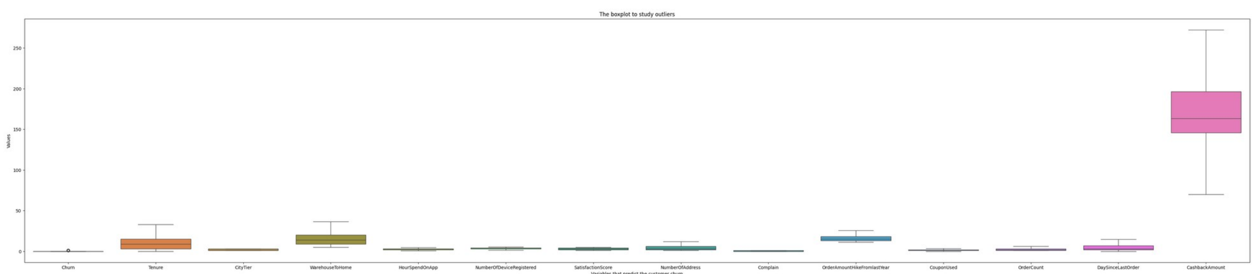

```
def remove_outlier(col):
    sorted(col)
    Q1,Q3=np.percentile(col,[25,75])
    IQR=Q3-Q1
    lr= Q1-(1.5 * IQR)
    ur= Q3+(1.5 * IQR)
    return lr, ur

df.columns
```

```
Index(['Churn', 'Tenure', 'PreferredLoginDevice', 'CityTier',
       'WarehouseToHome', 'PreferredPaymentMode', 'Gender',
'HourSpendOnApp',
       'NumberOfDeviceRegistered', 'PreferedOrderCat',
'SatisfactionScore',
```

```
       'MaritalStatus', 'NumberOfAddress', 'Complain',
       'OrderAmountHikeFromlastYear', 'CouponUsed', 'OrderCount',
       'DaySinceLastOrder', 'CashbackAmount'],
      dtype='object')
```

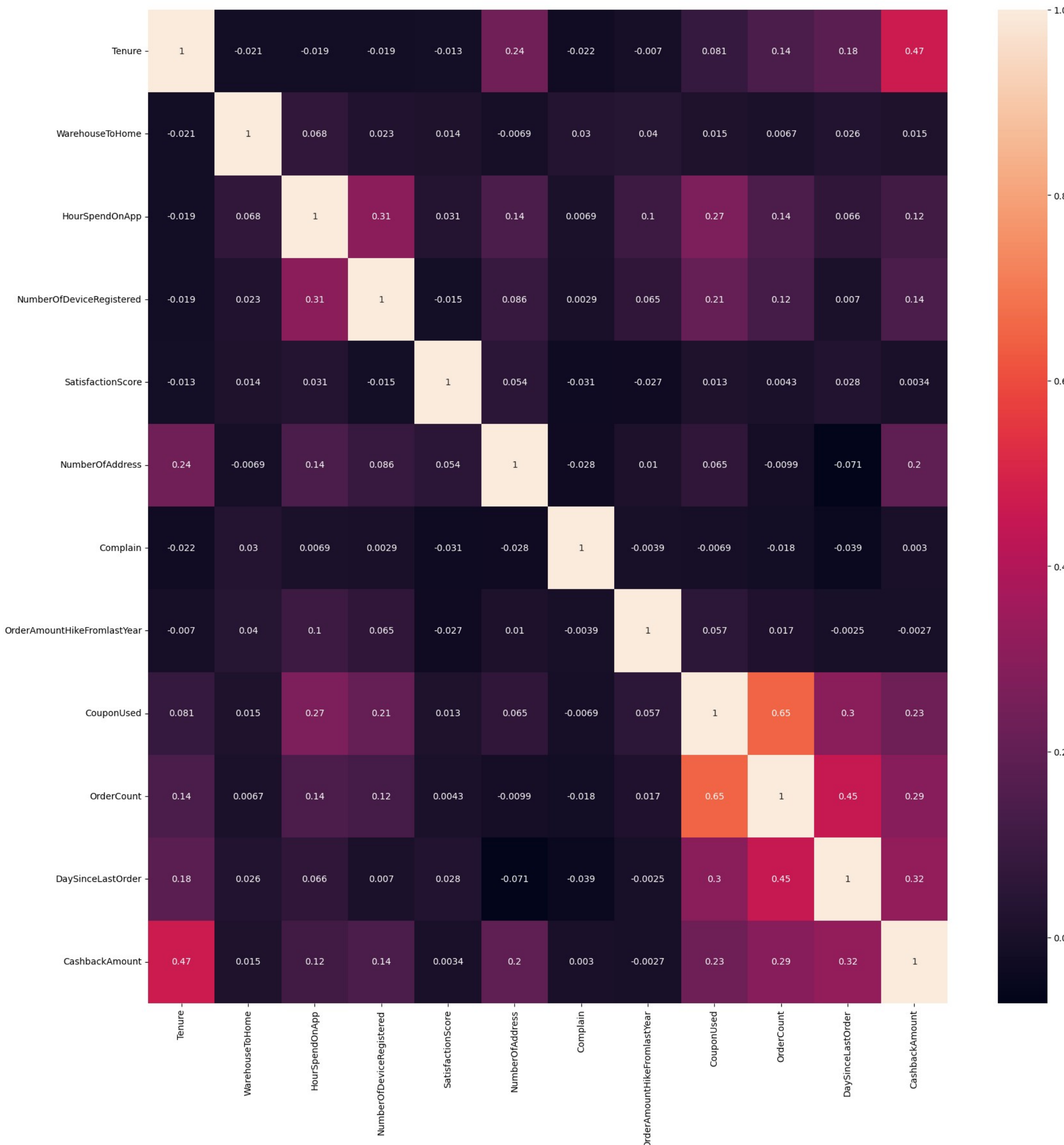
```
for column in df.columns:
    if df[column].dtype != 'object':
        lr,ur=remove_outlier(df[column])
        df[column]=np.where(df[column]>ur,ur,df[column])
        df[column]=np.where(df[column]<lr,lr,df[column])

plt.figure(figsize=(50,10))
sns.boxplot(data=df)
plt.title('The boxplot to study outliers')
plt.xlabel('Variables that predict the customer churn')
plt.ylabel('Values')
```

```
Text(0, 0.5, 'Values')
```




```
#heatmap for feature correlation
plt.figure(figsize=(20,20))
sns.heatmap(df.corr(),annot=True)
```

```
<ipython-input-16-974ef1362404>:3: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  sns.heatmap(df.corr(),annot=True)

<Axes: >
```




```
df_encoded=df.copy()
df_encoded.head()
```

{"summary":"{\n \"name\": \"df_encoded\",\n \"rows\": 5630,\n \"fields\": [\n   {\n   \"column\": \"Churn\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": 0,\n   \"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   0,\n   1\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Tenure\",\n   \"properties\": {\n   \"dtype\":

```
\"number\",\n        \"std\": 8.291333812302852,\n        \"min\":
0.0,\n        \"max\": 33.0,\n        \"num_unique_values\": 33,\n
\"samples\": [\n          33.0,\n          3.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"PreferredLoginDevice\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"Mobile
Phone\",\n          \"Phone\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"CityTier\",\n      \"properties\": {\n        \"dtype\":
\"date\",\n        \"min\": 1,\n        \"max\": 3,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          3,\n
1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"WarehouseToHome\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 8.089328252598381,\n        \"min\":
5.0,\n        \"max\": 36.5,\n        \"num_unique_values\": 33,\n
\"samples\": [\n          7.0,\n          14.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"PreferredPaymentMode\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 7,\n        \"samples\": [\n          \"Debit
Card\",\n          \"UPI\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Gender\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"Male\",\n          \"Female\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"HourSpendOnApp\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.7038263065618977,\n        \"min\": 0.5,\n        \"max\": 4.5,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          3.0,\n
2.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"NumberOfDeviceRegistered\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.9420153271843603,\n
\"min\": 1.5,\n        \"max\": 5.5,\n        \"num_unique_values\":
6,\n        \"samples\": [\n          3.0,\n          4.0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"PreferedOrderCat\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 6,\n        \"samples\": [\n          \"Laptop
& Accessory\",\n          \"Mobile\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"SatisfactionScore\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.38019445090948,\n        \"min\": 1.0,\n        \"max\": 5.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          3.0,\n
1.0\n        ],\n        \"semantic_type\": \"\",\n
```

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MaritalStatus\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"Single\",\n          \"Divorced\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"NumberOfAddress\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2.555110815760532,\n        \"min\": 1.0,\n        \"max\": 12.0,\n
\"num_unique_values\": 12,\n        \"samples\": [\n          12.0,\n
5.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Complain\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.4514079937386503,\n        \"min\":
0.0,\n        \"max\": 1.0,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          0.0,\n          1.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"OrderAmountHikeFromlastYear\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3.5828228430767135,\n        \"min\": 11.0,\n        \"max\": 25.5,\n
\"num_unique_values\": 16,\n        \"samples\": [\n          11.0,\n
15.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"CouponUsed\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1.0712954869323799,\n        \"min\":
0.0,\n        \"max\": 3.5,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.0,\n          3.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"OrderCount\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.7111087178674707,\n        \"min\": 1.0,\n        \"max\": 6.0,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          1.0,\n
6.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DaySinceLastOrder\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3.4391836447048068,\n        \"min\":
0.0,\n        \"max\": 14.5,\n        \"num_unique_values\": 16,\n
\"samples\": [\n          5.0,\n          0.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CashbackAmount\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
44.069816584596246,\n        \"min\": 69.83625,\n        \"max\":
272.32625,\n        \"num_unique_values\": 2365,\n        \"samples\":
[\n          134.82,\n          150.72\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df_encoded"}
```


```
cat=[]
num=[]
for i in df.columns:
  if df[i].dtype=='object':
```

```
    cat.append(i)
  else:
    num.append(i)
print('cat = ',cat)
print('num = ',num)
```

```
cat =  ['Churn', 'PreferredLoginDevice', 'CityTier',
'PreferredPaymentMode', 'Gender', 'PreferedOrderCat', 'MaritalStatus']
num =  ['Tenure', 'WarehouseToHome', 'HourSpendOnApp',
'NumberOfDeviceRegistered', 'SatisfactionScore', 'NumberOfAddress',
'Complain', 'OrderAmountHikeFromlastYear', 'CouponUsed', 'OrderCount',
'DaySinceLastOrder', 'CashbackAmount']
```

```
#encoding
df_encoded = pd.get_dummies(df_encoded,drop_first=True)
```

```
<ipython-input-19-0e5b7bbb6500>:2: FutureWarning: In a future version,
the Index constructor will not infer numeric dtypes when passed
object-dtype sequences (matching Series behavior)
  df_encoded = pd.get_dummies(df_encoded,drop_first=True)
<ipython-input-19-0e5b7bbb6500>:2: FutureWarning: In a future version,
the Index constructor will not infer numeric dtypes when passed
object-dtype sequences (matching Series behavior)
  df_encoded = pd.get_dummies(df_encoded,drop_first=True)
```

```
df_encoded.head(10)
```

{"type":"dataframe","variable_name":"df_encoded"}

```
from sklearn.preprocessing import StandardScaler
```

```
#standardisation
scaler = StandardScaler()

features = df_encoded[num]
features = scaler.fit_transform(features)

scaled_df_encoded = df_encoded.copy()
scaled_df_encoded[num] = features
scaled_df_encoded
```

{"type":"dataframe","variable_name":"scaled_df_encoded"}

```
print(scaled_df_encoded.columns)
```

```
Index(['Tenure', 'WarehouseToHome', 'HourSpendOnApp',
       'NumberOfDeviceRegistered', 'SatisfactionScore',
'NumberOfAddress',
       'Complain', 'OrderAmountHikeFromlastYear', 'CouponUsed',
'OrderCount',
       'DaySinceLastOrder', 'CashbackAmount', 'Churn_1',
```

```
       'PreferredLoginDevice_Mobile Phone',
'PreferredLoginDevice_Phone',
       'CityTier_2', 'CityTier_3', 'PreferredPaymentMode_COD',
       'PreferredPaymentMode_Cash on Delivery',
       'PreferredPaymentMode_Credit Card', 'PreferredPaymentMode_Debit
Card',
       'PreferredPaymentMode_E wallet', 'PreferredPaymentMode_UPI',
       'Gender_Male', 'PreferedOrderCat_Grocery',
       'PreferedOrderCat_Laptop & Accessory',
'PreferedOrderCat_Mobile',
       'PreferedOrderCat_Mobile Phone', 'PreferedOrderCat_Others',
       'MaritalStatus_Married', 'MaritalStatus_Single'],
      dtype='object')

from sklearn.model_selection import train_test_split

#train,test  split
X = scaled_df_encoded.drop(['Churn_1'], axis=1)
y = scaled_df_encoded['Churn_1']

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)

#lib for DL model building
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import regularizers
```

Baseline

```
# Baseline DNN Architecture
baseline_model = keras.Sequential([
    layers.Dense(128, activation='relu',
input_shape=(X_train.shape[1],)),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

baseline_model.compile(optimizer='adam',
                       loss='binary_crossentropy',
                       metrics=['accuracy'])

# Train the model
baseline_history = baseline_model.fit(X_train, y_train, epochs=20,
batch_size=32, validation_data=(X_val, y_val))

Epoch 1/20
141/141 [==============================] - 6s 14ms/step - loss: 0.3623
- accuracy: 0.8477 - val_loss: 0.2887 - val_accuracy: 0.8917
```

```
Epoch 2/20
141/141 [==============================] - 1s 9ms/step - loss: 0.2778
- accuracy: 0.8905 - val_loss: 0.2647 - val_accuracy: 0.8979
Epoch 3/20
141/141 [==============================] - 1s 8ms/step - loss: 0.2576
- accuracy: 0.8985 - val_loss: 0.2509 - val_accuracy: 0.9041
Epoch 4/20
141/141 [==============================] - 1s 8ms/step - loss: 0.2417
- accuracy: 0.9025 - val_loss: 0.2452 - val_accuracy: 0.9121
Epoch 5/20
141/141 [==============================] - 1s 8ms/step - loss: 0.2224
- accuracy: 0.9119 - val_loss: 0.2359 - val_accuracy: 0.9085
Epoch 6/20
141/141 [==============================] - 2s 11ms/step - loss: 0.2016
- accuracy: 0.9194 - val_loss: 0.2206 - val_accuracy: 0.9192
Epoch 7/20
141/141 [==============================] - 1s 6ms/step - loss: 0.1872
- accuracy: 0.9270 - val_loss: 0.2223 - val_accuracy: 0.9147
Epoch 8/20
141/141 [==============================] - 1s 4ms/step - loss: 0.1701
- accuracy: 0.9363 - val_loss: 0.1939 - val_accuracy: 0.9272
Epoch 9/20
141/141 [==============================] - 1s 4ms/step - loss: 0.1513
- accuracy: 0.9416 - val_loss: 0.1847 - val_accuracy: 0.9236
Epoch 10/20
141/141 [==============================] - 1s 4ms/step - loss: 0.1359
- accuracy: 0.9507 - val_loss: 0.1759 - val_accuracy: 0.9361
Epoch 11/20
141/141 [==============================] - 1s 4ms/step - loss: 0.1243
- accuracy: 0.9556 - val_loss: 0.1625 - val_accuracy: 0.9432
Epoch 12/20
141/141 [==============================] - 1s 4ms/step - loss: 0.1089
- accuracy: 0.9596 - val_loss: 0.1623 - val_accuracy: 0.9449
Epoch 13/20
141/141 [==============================] - 1s 4ms/step - loss: 0.0933
- accuracy: 0.9665 - val_loss: 0.1480 - val_accuracy: 0.9529
Epoch 14/20
141/141 [==============================] - 1s 4ms/step - loss: 0.0829
- accuracy: 0.9751 - val_loss: 0.1476 - val_accuracy: 0.9485
Epoch 15/20
141/141 [==============================] - 1s 6ms/step - loss: 0.0733
- accuracy: 0.9754 - val_loss: 0.1419 - val_accuracy: 0.9547
Epoch 16/20
141/141 [==============================] - 1s 5ms/step - loss: 0.0624
- accuracy: 0.9805 - val_loss: 0.1419 - val_accuracy: 0.9565
Epoch 17/20
141/141 [==============================] - 1s 5ms/step - loss: 0.0521
- accuracy: 0.9860 - val_loss: 0.1263 - val_accuracy: 0.9600
Epoch 18/20
```

```
141/141 [==============================] - 1s 4ms/step - loss: 0.0465
- accuracy: 0.9871 - val_loss: 0.1261 - val_accuracy: 0.9583
Epoch 19/20
141/141 [==============================] - 1s 4ms/step - loss: 0.0415
- accuracy: 0.9900 - val_loss: 0.1317 - val_accuracy: 0.9583
Epoch 20/20
141/141 [==============================] - 1s 4ms/step - loss: 0.0358
- accuracy: 0.9907 - val_loss: 0.1201 - val_accuracy: 0.9636
```

Dropout

```
# Dropout DNN Architecture
dropout_model = keras.Sequential([
    layers.Dense(128, activation='relu',
input_shape=(X_train.shape[1],)),
    layers.Dropout(0.5),#0.5 dropout rate
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.5),#0.5 dropout rate
    layers.Dense(1, activation='sigmoid')
])

dropout_model.compile(optimizer='adam',
                       loss='binary_crossentropy',
                       metrics=['accuracy'])

# Train the dropout model
dropout_history = dropout_model.fit(X_train, y_train, epochs=20,
batch_size=32, validation_data=(X_val, y_val))

Epoch 1/20
141/141 [==============================] - 3s 5ms/step - loss: 0.4521
- accuracy: 0.8113 - val_loss: 0.3265 - val_accuracy: 0.8668
Epoch 2/20
141/141 [==============================] - 1s 4ms/step - loss: 0.3430
- accuracy: 0.8595 - val_loss: 0.2912 - val_accuracy: 0.8908
Epoch 3/20
141/141 [==============================] - 1s 4ms/step - loss: 0.3203
- accuracy: 0.8752 - val_loss: 0.2776 - val_accuracy: 0.9014
Epoch 4/20
141/141 [==============================] - 1s 4ms/step - loss: 0.3116
- accuracy: 0.8777 - val_loss: 0.2705 - val_accuracy: 0.9023
Epoch 5/20
141/141 [==============================] - 1s 4ms/step - loss: 0.3025
- accuracy: 0.8830 - val_loss: 0.2625 - val_accuracy: 0.9050
Epoch 6/20
141/141 [==============================] - 1s 4ms/step - loss: 0.2893
- accuracy: 0.8819 - val_loss: 0.2595 - val_accuracy: 0.9023
Epoch 7/20
141/141 [==============================] - 1s 4ms/step - loss: 0.2786
- accuracy: 0.8899 - val_loss: 0.2534 - val_accuracy: 0.9147
```

```
Epoch 8/20
141/141 [==============================] - 1s 4ms/step - loss: 0.2733
- accuracy: 0.8914 - val_loss: 0.2483 - val_accuracy: 0.9139
Epoch 9/20
141/141 [==============================] - 1s 4ms/step - loss: 0.2676
- accuracy: 0.8965 - val_loss: 0.2438 - val_accuracy: 0.9174
Epoch 10/20
141/141 [==============================] - 1s 5ms/step - loss: 0.2631
- accuracy: 0.8970 - val_loss: 0.2456 - val_accuracy: 0.9165
Epoch 11/20
141/141 [==============================] - 1s 6ms/step - loss: 0.2576
- accuracy: 0.9034 - val_loss: 0.2398 - val_accuracy: 0.9165
Epoch 12/20
141/141 [==============================] - 1s 7ms/step - loss: 0.2550
- accuracy: 0.9010 - val_loss: 0.2370 - val_accuracy: 0.9192
Epoch 13/20
141/141 [==============================] - 1s 4ms/step - loss: 0.2396
- accuracy: 0.9067 - val_loss: 0.2341 - val_accuracy: 0.9227
Epoch 14/20
141/141 [==============================] - 1s 4ms/step - loss: 0.2423
- accuracy: 0.9019 - val_loss: 0.2334 - val_accuracy: 0.9192
Epoch 15/20
141/141 [==============================] - 1s 4ms/step - loss: 0.2317
- accuracy: 0.9130 - val_loss: 0.2243 - val_accuracy: 0.9192
Epoch 16/20
141/141 [==============================] - 1s 4ms/step - loss: 0.2320
- accuracy: 0.9079 - val_loss: 0.2284 - val_accuracy: 0.9210
Epoch 17/20
141/141 [==============================] - 1s 4ms/step - loss: 0.2309
- accuracy: 0.9107 - val_loss: 0.2194 - val_accuracy: 0.9272
Epoch 18/20
141/141 [==============================] - 1s 4ms/step - loss: 0.2227
- accuracy: 0.9143 - val_loss: 0.2152 - val_accuracy: 0.9272
Epoch 19/20
141/141 [==============================] - 1s 4ms/step - loss: 0.2221
- accuracy: 0.9145 - val_loss: 0.2206 - val_accuracy: 0.9183
Epoch 20/20
141/141 [==============================] - 1s 8ms/step - loss: 0.2177
- accuracy: 0.9141 - val_loss: 0.2194 - val_accuracy: 0.9272
```

Layer wise dropout

```
# Layer-wise Dropout DNN Architecture
layer_wise_dropout_model = keras.Sequential([
    layers.Dense(128, activation='relu',
input_shape=(X_train.shape[1],)),
    layers.Dropout(0.2),#0.2
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.3),#0.3
```

```
    layers.Dense(1, activation='sigmoid')
])

layer_wise_dropout_model.compile(optimizer='adam',
                                 loss='binary_crossentropy',
                                 metrics=['accuracy'])

# Train the layer-wise dropout model
layer_wise_dropout_history = layer_wise_dropout_model.fit(X_train,
y_train, epochs=20, batch_size=32, validation_data=(X_val, y_val))
```

```
Epoch 1/20
141/141 [==============================] - 5s 9ms/step - loss: 0.3849
- accuracy: 0.8408 - val_loss: 0.2920 - val_accuracy: 0.8908
Epoch 2/20
141/141 [==============================] - 1s 7ms/step - loss: 0.3015
- accuracy: 0.8777 - val_loss: 0.2697 - val_accuracy: 0.9005
Epoch 3/20
141/141 [==============================] - 1s 8ms/step - loss: 0.2906
- accuracy: 0.8803 - val_loss: 0.2651 - val_accuracy: 0.9023
Epoch 4/20
141/141 [==============================] - 1s 8ms/step - loss: 0.2776
- accuracy: 0.8914 - val_loss: 0.2491 - val_accuracy: 0.9112
Epoch 5/20
141/141 [==============================] - 1s 8ms/step - loss: 0.2626
- accuracy: 0.8950 - val_loss: 0.2430 - val_accuracy: 0.9112
Epoch 6/20
141/141 [==============================] - 1s 7ms/step - loss: 0.2550
- accuracy: 0.8979 - val_loss: 0.2395 - val_accuracy: 0.9165
Epoch 7/20
141/141 [==============================] - 1s 9ms/step - loss: 0.2448
- accuracy: 0.9036 - val_loss: 0.2319 - val_accuracy: 0.9139
Epoch 8/20
141/141 [==============================] - 1s 9ms/step - loss: 0.2362
- accuracy: 0.9072 - val_loss: 0.2328 - val_accuracy: 0.9192
Epoch 9/20
141/141 [==============================] - 1s 9ms/step - loss: 0.2210
- accuracy: 0.9083 - val_loss: 0.2290 - val_accuracy: 0.9227
Epoch 10/20
141/141 [==============================] - 1s 8ms/step - loss: 0.2113
- accuracy: 0.9181 - val_loss: 0.2206 - val_accuracy: 0.9254
Epoch 11/20
141/141 [==============================] - 1s 7ms/step - loss: 0.2042
- accuracy: 0.9194 - val_loss: 0.2131 - val_accuracy: 0.9272
Epoch 12/20
141/141 [==============================] - 1s 7ms/step - loss: 0.1962
- accuracy: 0.9254 - val_loss: 0.2066 - val_accuracy: 0.9281
Epoch 13/20
141/141 [==============================] - 1s 7ms/step - loss: 0.1867
- accuracy: 0.9274 - val_loss: 0.2046 - val_accuracy: 0.9272
```

```
Epoch 14/20
141/141 [==============================] - 1s 7ms/step - loss: 0.1763
- accuracy: 0.9325 - val_loss: 0.1916 - val_accuracy: 0.9343
Epoch 15/20
141/141 [==============================] - 1s 5ms/step - loss: 0.1706
- accuracy: 0.9343 - val_loss: 0.1966 - val_accuracy: 0.9298
Epoch 16/20
141/141 [==============================] - 1s 4ms/step - loss: 0.1683
- accuracy: 0.9349 - val_loss: 0.1860 - val_accuracy: 0.9343
Epoch 17/20
141/141 [==============================] - 1s 4ms/step - loss: 0.1551
- accuracy: 0.9398 - val_loss: 0.1941 - val_accuracy: 0.9361
Epoch 18/20
141/141 [==============================] - 1s 4ms/step - loss: 0.1530
- accuracy: 0.9356 - val_loss: 0.1842 - val_accuracy: 0.9361
Epoch 19/20
141/141 [==============================] - 1s 4ms/step - loss: 0.1428
- accuracy: 0.9460 - val_loss: 0.1739 - val_accuracy: 0.9396
Epoch 20/20
141/141 [==============================] - 1s 4ms/step - loss: 0.1383
- accuracy: 0.9458 - val_loss: 0.1815 - val_accuracy: 0.9352
```

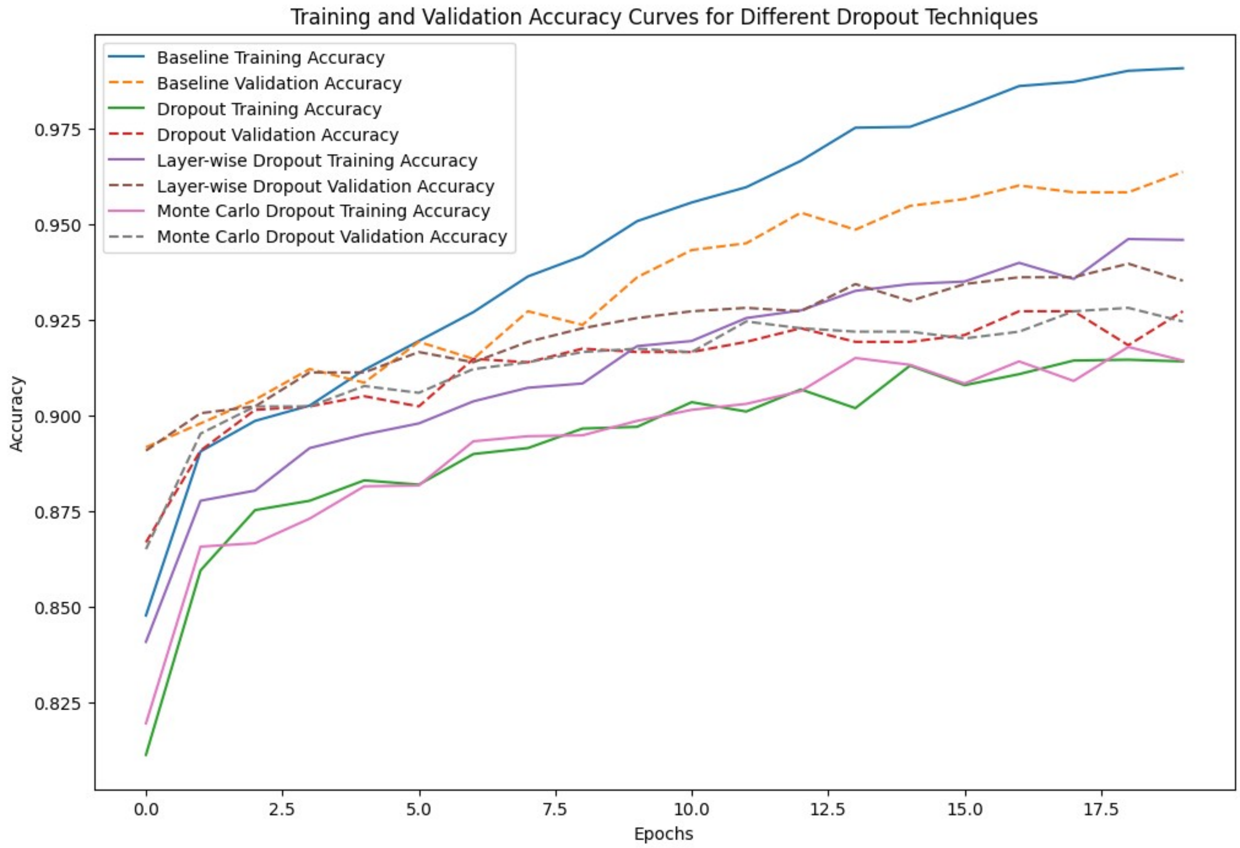
Monte Carlo dropout

```
# Monte Carlo Dropout DNN Architecture
monte_carlo_dropout_model = keras.Sequential([
    layers.Dense(128, activation='relu',
input_shape=(X_train.shape[1],)),
    layers.Dropout(0.5),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(1, activation='sigmoid')
])

monte_carlo_dropout_model.compile(optimizer='adam',
                                  loss='binary_crossentropy',
                                  metrics=['accuracy'])

monte_carlo_dropout_history = monte_carlo_dropout_model.fit(X_train,
y_train, epochs=20, batch_size=32, validation_data=(X_val, y_val))

Epoch 1/20
141/141 [==============================] - 2s 8ms/step - loss: 0.4205
- accuracy: 0.8195 - val_loss: 0.3139 - val_accuracy: 0.8650
Epoch 2/20
141/141 [==============================] - 1s 6ms/step - loss: 0.3284
- accuracy: 0.8657 - val_loss: 0.2867 - val_accuracy: 0.8952
Epoch 3/20
141/141 [==============================] - 1s 4ms/step - loss: 0.3146
- accuracy: 0.8666 - val_loss: 0.2768 - val_accuracy: 0.9023
```

```
Epoch 4/20
141/141 [==============================] - 1s 4ms/step - loss: 0.3016
- accuracy: 0.8730 - val_loss: 0.2666 - val_accuracy: 0.9023
Epoch 5/20
141/141 [==============================] - 1s 4ms/step - loss: 0.2907
- accuracy: 0.8814 - val_loss: 0.2594 - val_accuracy: 0.9076
Epoch 6/20
141/141 [==============================] - 1s 4ms/step - loss: 0.2860
- accuracy: 0.8817 - val_loss: 0.2596 - val_accuracy: 0.9059
Epoch 7/20
141/141 [==============================] - 1s 4ms/step - loss: 0.2710
- accuracy: 0.8932 - val_loss: 0.2499 - val_accuracy: 0.9121
Epoch 8/20
141/141 [==============================] - 1s 4ms/step - loss: 0.2672
- accuracy: 0.8945 - val_loss: 0.2481 - val_accuracy: 0.9139
Epoch 9/20
141/141 [==============================] - 1s 4ms/step - loss: 0.2635
- accuracy: 0.8948 - val_loss: 0.2433 - val_accuracy: 0.9165
Epoch 10/20
141/141 [==============================] - 1s 4ms/step - loss: 0.2587
- accuracy: 0.8985 - val_loss: 0.2372 - val_accuracy: 0.9174
Epoch 11/20
141/141 [==============================] - 1s 4ms/step - loss: 0.2482
- accuracy: 0.9014 - val_loss: 0.2339 - val_accuracy: 0.9165
Epoch 12/20
141/141 [==============================] - 1s 7ms/step - loss: 0.2518
- accuracy: 0.9030 - val_loss: 0.2335 - val_accuracy: 0.9245
Epoch 13/20
141/141 [==============================] - 1s 7ms/step - loss: 0.2334
- accuracy: 0.9063 - val_loss: 0.2254 - val_accuracy: 0.9227
Epoch 14/20
141/141 [==============================] - 1s 5ms/step - loss: 0.2292
- accuracy: 0.9150 - val_loss: 0.2283 - val_accuracy: 0.9218
Epoch 15/20
141/141 [==============================] - 1s 7ms/step - loss: 0.2288
- accuracy: 0.9132 - val_loss: 0.2244 - val_accuracy: 0.9218
Epoch 16/20
141/141 [==============================] - 1s 5ms/step - loss: 0.2250
- accuracy: 0.9083 - val_loss: 0.2229 - val_accuracy: 0.9201
Epoch 17/20
141/141 [==============================] - 1s 9ms/step - loss: 0.2214
- accuracy: 0.9141 - val_loss: 0.2144 - val_accuracy: 0.9218
Epoch 18/20
141/141 [==============================] - 1s 8ms/step - loss: 0.2246
- accuracy: 0.9090 - val_loss: 0.2105 - val_accuracy: 0.9272
Epoch 19/20
141/141 [==============================] - 1s 8ms/step - loss: 0.2117
- accuracy: 0.9179 - val_loss: 0.2094 - val_accuracy: 0.9281
Epoch 20/20
```

```
141/141 [==============================] - 1s 6ms/step - loss: 0.2073
- accuracy: 0.9143 - val_loss: 0.2055 - val_accuracy: 0.9245
```

```
# Visualize training and validation accuracy/loss curves for all
models
plt.figure(figsize=(12, 8))

# Baseline model
plt.plot(baseline_history.history['accuracy'], label='Baseline
Training Accuracy', linestyle='-')
plt.plot(baseline_history.history['val_accuracy'], label='Baseline
Validation Accuracy', linestyle='--')

# Dropout model
plt.plot(dropout_history.history['accuracy'], label='Dropout Training
Accuracy', linestyle='-')
plt.plot(dropout_history.history['val_accuracy'], label='Dropout
Validation Accuracy', linestyle='--')

# Layer-wise dropout model
plt.plot(layer_wise_dropout_history.history['accuracy'], label='Layer-
wise Dropout Training Accuracy', linestyle='-')
plt.plot(layer_wise_dropout_history.history['val_accuracy'],
label='Layer-wise Dropout Validation Accuracy', linestyle='--')

# Monte Carlo dropout model
plt.plot(monte_carlo_dropout_history.history['accuracy'], label='Monte
Carlo Dropout Training Accuracy', linestyle='-')
plt.plot(monte_carlo_dropout_history.history['val_accuracy'],
label='Monte Carlo Dropout Validation Accuracy', linestyle='--')

plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy Curves for Different
Dropout Techniques')
plt.legend()
plt.show()
```




```
# metrics
from sklearn.metrics import accuracy_score, f1_score, roc_curve,
roc_auc_score

# Predictions for each model
baseline_probs = baseline_model.predict(X_val)
dropout_probs = dropout_model.predict(X_val)
layer_wise_dropout_probs = layer_wise_dropout_model.predict(X_val)
monte_carlo_dropout_probs = monte_carlo_dropout_model.predict(X_val)

# Convert probabilities to binary predictions
baseline_preds = (baseline_probs > 0.5).astype(int)
dropout_preds = (dropout_probs > 0.5).astype(int)
layer_wise_dropout_preds = (layer_wise_dropout_probs >
0.5).astype(int)
monte_carlo_dropout_preds = (monte_carlo_dropout_probs >
0.5).astype(int)

# Accuracy scores
baseline_accuracy = accuracy_score(y_val, baseline_preds)
dropout_accuracy = accuracy_score(y_val, dropout_preds)
layer_wise_dropout_accuracy = accuracy_score(y_val,
layer_wise_dropout_preds)
monte_carlo_dropout_accuracy = accuracy_score(y_val,
```

```
monte_carlo_dropout_preds)

# F1 scores
baseline_f1 = f1_score(y_val, baseline_preds)
dropout_f1 = f1_score(y_val, dropout_preds)
layer_wise_dropout_f1 = f1_score(y_val, layer_wise_dropout_preds)
monte_carlo_dropout_f1 = f1_score(y_val, monte_carlo_dropout_preds)

# ROC curves and AUC scores
baseline_auc = roc_auc_score(y_val, baseline_preds)
dropout_auc = roc_auc_score(y_val, dropout_preds)
layer_wise_dropout_auc = roc_auc_score(y_val,
layer_wise_dropout_preds)
monte_carlo_dropout_auc = roc_auc_score(y_val,
monte_carlo_dropout_preds)
```

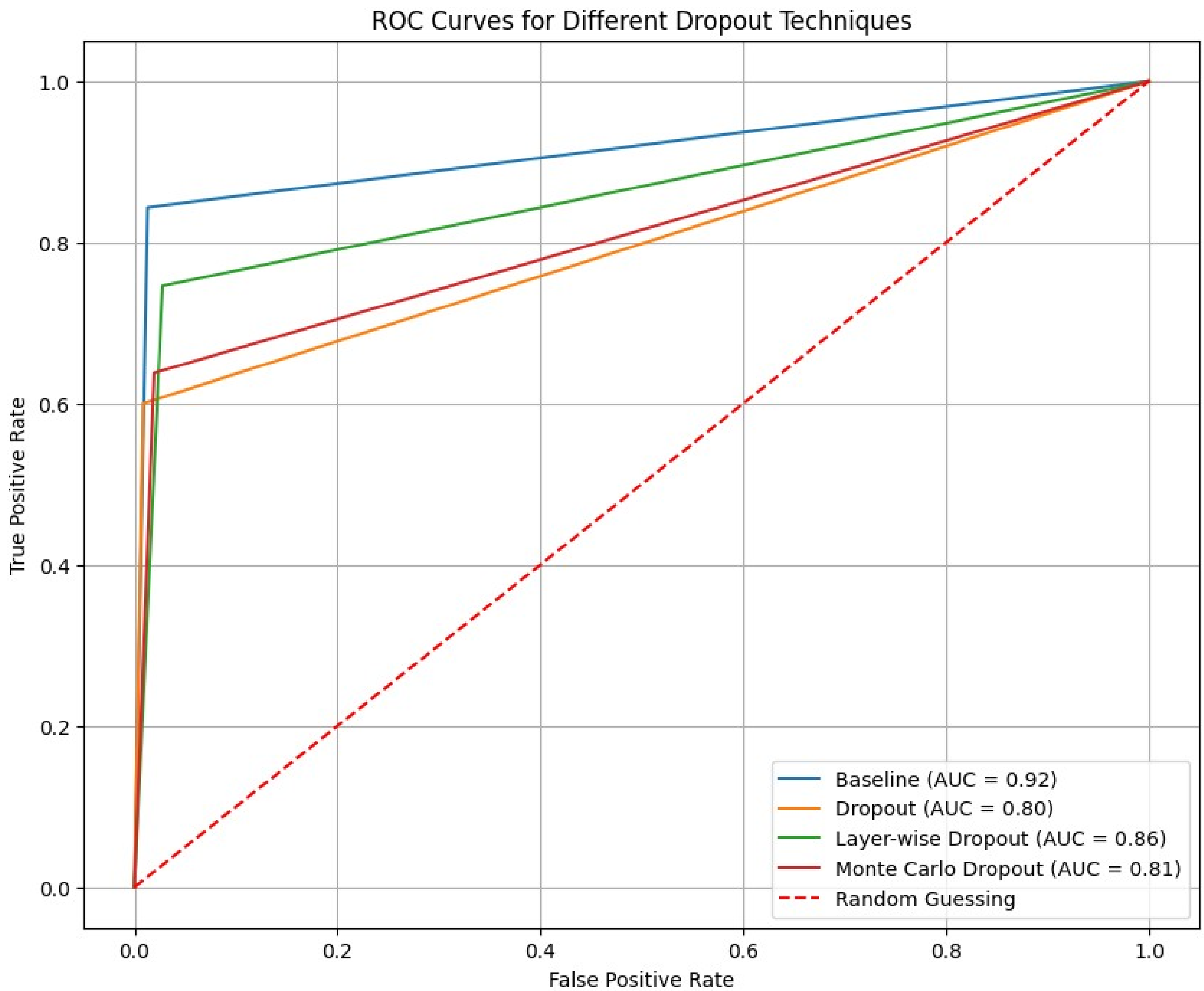
```
36/36 [==============================] - 0s 3ms/step
36/36 [==============================] - 0s 2ms/step
36/36 [==============================] - 0s 3ms/step
36/36 [==============================] - 0s 2ms/step
```

```
# Print performance metrics
print("Accuracy Scores:")
print(f"Baseline: {baseline_accuracy:.4f}")
print(f"With Dropout: {dropout_accuracy:.4f}")
print(f"With Layer-wise Dropout: {layer_wise_dropout_accuracy:.4f}")
print(f"With Monte Carlo Dropout: {monte_carlo_dropout_accuracy:.4f}\
n")

print("F1 Scores:")
print(f"Baseline: {baseline_f1:.4f}")
print(f"With Dropout: {dropout_f1:.4f}")
print(f"With Layer-wise Dropout: {layer_wise_dropout_f1:.4f}")
print(f"With Monte Carlo Dropout: {monte_carlo_dropout_f1:.4f}\n")

print("ROC AUC Scores:")
print(f"Baseline: {baseline_auc:.4f}")
print(f"With Dropout: {dropout_auc:.4f}")
print(f"With Layer-wise Dropout: {layer_wise_dropout_auc:.4f}")
print(f"With Monte Carlo Dropout: {monte_carlo_dropout_auc:.4f}\n")
```

```
Accuracy Scores:
Baseline: 0.9636
With Dropout: 0.9272
With Layer-wise Dropout: 0.9352
With Monte Carlo Dropout: 0.9245

F1 Scores:
Baseline: 0.8839
With Dropout: 0.7303
```

```
With Layer-wise Dropout: 0.7908
With Monte Carlo Dropout: 0.7352

ROC AUC Scores:
Baseline: 0.9152
With Dropout: 0.7957
With Layer-wise Dropout: 0.8592
With Monte Carlo Dropout: 0.8094
```

```python
# Plot ROC curves
plt.figure(figsize=(10, 8))
fpr, tpr, _ = roc_curve(y_val, baseline_preds)
plt.plot(fpr, tpr, label=f'Baseline (AUC = {baseline_auc:.2f})')

fpr, tpr, _ = roc_curve(y_val, dropout_preds)
plt.plot(fpr, tpr, label=f'Dropout (AUC = {dropout_auc:.2f})')

fpr, tpr, _ = roc_curve(y_val, layer_wise_dropout_preds)
plt.plot(fpr, tpr, label=f'Layer-wise Dropout (AUC =
{layer_wise_dropout_auc:.2f})')

fpr, tpr, _ = roc_curve(y_val, monte_carlo_dropout_preds)
plt.plot(fpr, tpr, label=f'Monte Carlo Dropout (AUC =
{monte_carlo_dropout_auc:.2f})')

plt.plot([0, 1], [0, 1], linestyle='--', color='r', label='Random
Guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for Different Dropout Techniques')
plt.legend()
plt.grid()
plt.show()
```


ROC Curves for Different Dropout Techniques
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Baseline (AUC = 0.92)
Dropout (AUC = 0.80)
Layer-wise Dropout (AUC = 0.86)
Monte Carlo Dropout (AUC = 0.81)
Random Guessing