```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import StringType,StructType
,IntegerType,DoubleType,StructField

spark = SparkSession.builder.master("local[*]").
getOrCreate()

df = spark.read.csv(r"D:\FINALPROJECTS\New folder\
finaldata2.csv",header=True)
df.printSchema()
df.show()

df1 = df.select("patientid","patient_name","
date_of_birth","age","gender","address","city","
contact_number")
df1.show()


print("################### INSURANCE PROVIDER AND
PLAN ###################")


df1 = spark.read.csv(r"D:\FINALPROJECTS\New folder\
BLUECROSS_OSCARHEALT_AETNA_CIGNA(
INSURANCE_PROVIDERS).csv",header=True)
df1.printSchema()
df1.show()

print("records are",df1.count())


print("################### PATIENT INFO
 ###################")

df2 = spark.read.csv(r"D:\FINALPROJECTS\New folder\
CERNER_CUREMD_PRACTO(PATIENT_INFO).csv",header=True
)
```

```
df21 = df2.select("patientid","patient_name","
date_of_birth","p_age","gender","patient_address","
city","contact_number")
df21.printSchema()
df21.show()



print("################## CLAIM DATA
 ##################")


df3 = spark.read.csv(r"D:\FINALPROJECTS\New folder\
CIGNA(INSURANCE_CLAIM_DATA).csv",header=True)
df3.printSchema()
df3.show()



print("################## PAYMENT INFO
 ##################")


df4 = spark.read.csv(r"D:\FINALPROJECTS\New folder\
CREDIT_UPI_PAYZAPP(PAYMENT_INFO).csv",header=True)
df4.printSchema()
df4.show()



print("################## DISEASE INFO
 ##################")


df5 = spark.read.csv(r"D:\FINALPROJECTS\New folder\
HEALTHTAP_HEALTHVAULT(patient_information).csv",
header=True)
# df5= df5.select("patientid","hospital_id","
```

```
DISEASE","SUB_DISEASE","DISEASE_CODE","DIAGNOSIS/
TREATMENT","MEDICINE")
df5.printSchema()
df5.show()



print("################## MEDICINE INFO
 ##################")


df6 = spark.read.csv(r"D:\FINALPROJECTS\New folder\
MEDISAFE_CAREZONE_GOODRX_NETMEDS(MEDICINE_BILLS).
csv",header=True)
df6= df6.select("patientid","hospital_id","
physician_id","DISEASE_CODE","medicine_names","
medical_bill_AMOUNT")

df6.printSchema()
df6.show()



print("################## PROVIDER INFO
 ##################")


df7 = spark.read.csv(r"D:\FINALPROJECTS\New folder\
HOSPITALS.csv",header=True)
df7 = df7.select("hospital_id","Hospital_address","
physician_id","Speciality","INSURANCE_provider_name
","insurance_provider_id")
df7.printSchema()
df7.show()
```

```
print("################## JOIN 1
 ##################")
print("PATIENT_INFO,DISEASE_INFO,MEDICINE_INFO")
df8 = df2.join(df5,"patientid","inner")\
        .join(df6,"patientid","inner")
df8.printSchema()
df8.show()

print("Count of above dataframe is : - ",df8.count
())



print("################## JOIN 2
 ##################")
print("HOSPITALS_INFO,CLAIM_INFO,PAYMENT_INFO")


df9 = df3.join(df4,"hospital_id","inner")\
        .join(df7,"hospital_id","inner")
df9.printSchema()
df9.show()

print("Count of above dataframe is : - ",df9.count
())



print("################## COMBINED_TABLES
##################")

df10 = df8.join(df9,"hospital_id","inner")\
            .join(df1,"health_insurance_id","inner"
)

df10.printSchema()
```

```
df10.show()

print("Count of above dataframe is : - ",df10.count
())



print("################## SELECTED COLUMNS FROM
TABLES###################")


df11 = df10.select("patientid",df21.patient_name,
df21.date_of_birth,df21.p_age,df21.gender,df21.
patient_address,df3.health_insurance_id,\
                   df3.hospital_id,"
Hospital_address",df7.physician_id,df7.Speciality,
df7.INSURANCE_provider_name,df7.
insurance_provider_id,\
                   df5.DISEASE,df5.SUB_DISEASE,df5.
DISEASE_CODE,df5["DIAGNOSIS/TREATMENT"],df6.
medicine_names,df6.medical_bill_AMOUNT,"
hospital_bill_AMOUNT",\
                   "INSURANCE_PLAN","
INSURANCE_PREMIUM","insurance_start_date","
insurance_end_date",df3.claim_id,df3.claim_amount,
df3.deductible,\
                   df3.copay,df3.COINSURANCE,df3.
outofpocketmax,df3.claim_amount_settled,"
payment_status","payment_method","payment_id")

df12 = df11.coalesce(1)

df12.write.csv(r"D://FINALPROJECTS/New folder/
final_data/",mode="overwrite",header=True)

print("File successfully merged and saved to local
system")
```

```
print("############################   ADVANCED
TRANSFORMATIONS ##############################")


### Problem 1: Fraud Detection

# Identified fraudulent claims based on patterns
such as unusually high claim amounts or frequent
claims from the same provider.

schema1 = StructType([
    StructField("patientid", StringType(), True),
    StructField("patient_name", StringType(), True
),
    StructField("date_of_birth", StringType(), True
),
    StructField("p_age", IntegerType(), True),
    StructField("gender", StringType(), True),
    StructField("patient_address", StringType(),
True),
    StructField("health_insurance_id", StringType
(), True),
    StructField("hospital_id", StringType(), True),
    StructField("Hospital_address", StringType(),
True),
    StructField("physician_id", StringType(), True
),
    StructField("Speciality", StringType(), True),
    StructField("INSURANCE_provider_name",
StringType(), True),
    StructField("insurance_provider_id", StringType
(), True),
    StructField("DISEASE", StringType(), True),
    StructField("SUB_DISEASE", StringType(), True),
    StructField("DISEASE_CODE", StringType(), True
),
    StructField("DIAGNOSIS/TREATMENT", StringType
```

```
(), True),
    StructField("medicine_names", StringType(),
True),
    StructField("medical_bill_AMOUNT", IntegerType
(), True),
    StructField("hospital_bill_AMOUNT", IntegerType
(), True),
    StructField("INSURANCE_PLAN", StringType(),
True),
    StructField("INSURANCE_PREMIUM", IntegerType
(), True),
    StructField("insurance_start_date", StringType
(), True),
    StructField("insurance_end_date", StringType
(), True),
    StructField("claim_id", StringType(), True),
    StructField("claim_amount", IntegerType(), True
),
    StructField("deductible", IntegerType(), True),
    StructField("copay", IntegerType(), True),
    StructField("COINSURANCE", DoubleType(), True),
    StructField("outofpocketmax", DoubleType(),
True),
    StructField("claim_amount_settled", DoubleType
(), True),
    StructField("payment_status", StringType(),
True),
    StructField("payment_method", StringType(),
True),
    StructField("payment_id", StringType(), True)
])

df20 = spark.read.csv(r"D:\FINALPROJECTS\New folder
\merged_data\merged_data_final - Copy.csv",header=
True,schema=schema1)
df20.printSchema()

df20 = df20.select("patientid","hospital_id","p_age
```

```
","DISEASE","insurance_provider_id","
medical_bill_AMOUNT",\
                    "hospital_bill_AMOUNT","
INSURANCE_PLAN","claim_id","claim_amount","
deductible",\
                    "copay","COINSURANCE","
outofpocketmax","claim_amount_settled")
# Define thresholds
high_claim_threshold = 4200
frequent_claim_threshold = 2

# Identify high-value claims
fraudulent_claims = df20.withColumn("
high_value_claim", when(df20.claim_amount_settled
 > high_claim_threshold, "Yes").otherwise("No"))

# Identify frequent claims
frequent_claims = df20.groupBy("hospital_id").count
().withColumnRenamed("count", "claim_count")
frequent_claims = frequent_claims.withColumn("
frequent_claims", when(col("claim_count") >
frequent_claim_threshold, "Yes").otherwise("No"))

# Join datasets to get potentially fraudulent
claims
fraudulent_claims = fraudulent_claims.join(
frequent_claims, on="hospital_id", how="left").
filter((col("high_value_claim") == "Yes") | (col("
frequent_claims") == "Yes"))

# Order by high_value_claim in descending order
fraudulent_claims = fraudulent_claims.orderBy(col("
high_value_claim").desc())

# Show the schema and top 20 rows of the result
fraudulent_claims.printSchema()
fraudulent_claims.show(truncate=False)
```

```
print("Count of claims which are greater than
threshold claim amount :-",fraudulent_claims.count
())


print(
"##################################################
#################################")

print("#################### Hospital wise fraud
detection ##################### ")


frauds = fraudulent_claims.withColumn("
fraud_detected", when(((col("high_value_claim") ==
"Yes") & (col("frequent_claims") == "Yes")), "Yes"
).otherwise("No"))

frauds = frauds.select("hospital_id","claim_id","
insurance_provider_id","claim_amount_settled","
high_value_claim","claim_count","frequent_claims",
col("fraud_detected"))

frauds = frauds.filter((col("high_value_claim") ==
"Yes") & (col("frequent_claims") == "Yes"))

frauds.show()

print("Count of fraud claims are :-",frauds.count
())


print("#################### Disease wise fraud
detection ##################### ")

print("########### CANCER ###########")

# Defined thresholds for cancer
```

```
high_claim_threshold = 4299
frequent_claim_threshold = 2

# Identifying high-value claims

fraudulent_claims = df20.withColumn("
high_value_claim", when(df20.claim_amount_settled
 > high_claim_threshold, "Yes").otherwise("No"))

# Identifying frequent claims

frequent_claims = df20.groupBy("hospital_id").count
().withColumnRenamed("count", "claim_count")

frequent_claims = frequent_claims.withColumn("
frequent_claims", when(col("claim_count") >
frequent_claim_threshold, "Yes").otherwise("No"))

# Join datasets to get potentially fraudulent
claims
fraudulent_claims = fraudulent_claims.join(
frequent_claims, on="hospital_id", how="left").
filter((col("high_value_claim") == "Yes") | (col("
frequent_claims") == "Yes"))

# Order by high_value_claim in descending order
fraudulent_claims = fraudulent_claims.filter(col("
disease")=="Cancer").orderBy(col("high_value_claim"
).desc())

fraudulent_claims.show()

print("Count of claims which are greater than
threshold cancer claim amount :-",fraudulent_claims
.count())


frauds = fraudulent_claims.withColumn("
```

```
fraud_detected", when(((col("high_value_claim") ==
"Yes") & (col("frequent_claims") == "Yes")), "Yes"
).otherwise("No"))

frauds = frauds.select("hospital_id","claim_id","
insurance_provider_id","claim_amount_settled","
high_value_claim","claim_count","frequent_claims",
col("fraud_detected"))

frauds = frauds.filter((col("high_value_claim") ==
"Yes") & (col("frequent_claims") == "Yes"))

frauds.show()

print("Count of fraud claims are :-",frauds.count
())



print("##################### Cost Analysis
 ##################### ")

### Problem 2: Cost Analysis

# Analyzed the cost distribution among different
types of claims and identify the top contributors
to the overall cost.



# Calculate total cost for each claim type

cost_analysis = df20.groupBy("INSURANCE_PLAN").agg(
    sum("medical_bill_AMOUNT").alias("
total_medical_bill"),
    sum("hospital_bill_AMOUNT").alias("
total_hospital_bill"),
    sum("copay").alias("total_copay"),
    sum("coinsurance").alias("total_coinsurance"),
    sum("deductible").alias("total_deductible"),
```

```
    sum("claim_amount").alias("total_claim_amount")
)

# Calculate percentage contribution of each type

total_claims = cost_analysis.agg(sum("
total_claim_amount").alias("total")).collect()[0]["
total"]
cost_analysis = cost_analysis.withColumn("
percentage_contribution", col("total_claim_amount"
) / total_claims * 100)

cost_analysis.orderBy(col("total_claim_amount").
desc()).show()


print("##################### Patient Cost Burden
Analysis ###################### ")


### Problem 3: Patient Cost Burden Analysis

# Analyzed the out-of-pocket cost burden on
patients, considering copay, coinsurance, and
deductibles.

# Calculating patient out-of-pocket costs

# patient_cost_burden = df20.withColumn("
out_of_pocket", col("copay") + col("coinsurance
") + col("deductible"))


average_cost_burden = df20.groupBy("patientid").agg
(
    round(sum("outofpocketmax") / count("claim_id"
), 2).alias("avg_out_of_pocket"))
```

```
# patients with highest average out-of-pocket costs

average_cost_burden.orderBy(col("avg_out_of_pocket"
).desc()).show()



print("##################### Provider Performance
Analysis ##################### ")


### Problem 4: Provider Performance Analysis

# Calculate average claim amount and frequency of
claims for each provider
provider_performance = df20.groupBy("
insurance_provider_id").agg(
    avg("claim_amount_settled").alias("
avg_claim_amount"),
    count("claim_id").alias("claim_count")
)

# Providers with highest average claim amount and
frequency
provider_performance.orderBy(col("avg_claim_amount"
).desc(), col("claim_count").desc()).show()


### Problem 6: Calculate total coverage provided by
 each insurance plan


insurance_coverage = df20.groupBy("INSURANCE_PLAN"
).agg(
    avg("claim_amount_settled").alias("
avg_claim_amount_settled"),
    max("claim_amount_settled").alias("
max_claim_amount_settled"),
```

```
    min("claim_amount_settled").alias("
min_claim_amount_settled")
)


print("Plan with total coverages :-",
insurance_coverage.show())


print("#################### Age groupwise analysis
 ################### ")


### Problem 7: Age groupwise analysis

df20 = df20.withColumn("age_group", when(col("p_age
") < 18, "Child")
                                    .when((col("
p_age") >= 18) & (col("p_age") < 60), "Adult")
                                    .otherwise("
Senior"))

# Calculate average ,max,min claim amounts by age
group
age_group_analysis = df20.groupBy("age_group").agg(
    avg("claim_amount").alias("avg_claim_amount"),
    max("claim_amount").alias("max_claim_amount"),
    min("claim_amount").alias("min_claim_amount")
)

# Show the result
age_group_analysis.show()



### Problem 6: Claim Processing Time Analysis

# Assuming 'claim_submission_date' and '
```

```
claim_processing_date' are present in the data
# data = df20.withColumn("processing_time",
datediff(col("claim_processing_date"), col("
claim_submission_date")))
#
# # Calculate average processing time per provider
# processing_time_analysis = data.groupBy("
provider_id").agg(
#     avg("processing_time").alias("
avg_processing_time")
# )
#
# providers with the longest average processing
times
# processing_time_analysis.orderBy(col("
avg_processing_time").desc()).show()
```