1. Load data and create a Spark data frame

```
scala>  val df = spark.read.option("header",true).option("inferSchema",true).csv("E:/BIG DATA/market_analysis_in_banking_domain.csv")

df: org.apache.spark.sql.DataFrame = [age: int, job: string ... 15 more fields]

scala> df.show()
```

Command Prompt - spark-shell

```
scala> df.show()
+---+------------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+---+
|age|         job| marital|education|default|balance|housing|loan|contact|day|month|duration|campaign|pdays|previous|poutcome|  y|
+---+------------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+---+
| 58|  management| married| tertiary|     no|   2143|    yes|  no|unknown|  5|  may|     261|       1|   -1|       0| unknown| no|
| 44|  technician|  single|secondary|     no|     29|    yes|  no|unknown|  5|  may|     151|       1|   -1|       0| unknown| no|
| 33|entrepreneur| married|secondary|     no|      2|    yes| yes|unknown|  5|  may|      76|       1|   -1|       0| unknown| no|
| 47| blue-collar| married|  unknown|     no|   1506|    yes|  no|unknown|  5|  may|      92|       1|   -1|       0| unknown| no|
| 33|     unknown|  single|  unknown|     no|      1|     no|  no|unknown|  5|  may|     198|       1|   -1|       0| unknown| no|
| 35|  management| married| tertiary|     no|    231|    yes|  no|unknown|  5|  may|     139|       1|   -1|       0| unknown| no|
| 28|  management|  single| tertiary|     no|    447|    yes| yes|unknown|  5|  may|     217|       1|   -1|       0| unknown| no|
| 42|entrepreneur|divorced| tertiary|    yes|      2|    yes|  no|unknown|  5|  may|     380|       1|   -1|       0| unknown| no|
| 58|     retired| married|  primary|     no|    121|    yes|  no|unknown|  5|  may|      50|       1|   -1|       0| unknown| no|
| 43|  technician|  single|secondary|     no|    593|    yes|  no|unknown|  5|  may|      55|       1|   -1|       0| unknown| no|
| 41|      admin.|divorced|secondary|     no|    270|    yes|  no|unknown|  5|  may|     222|       1|   -1|       0| unknown| no|
| 29|      admin.|  single|secondary|     no|    390|    yes|  no|unknown|  5|  may|     137|       1|   -1|       0| unknown| no|
| 53|  technician| married|secondary|     no|      6|    yes|  no|unknown|  5|  may|     517|       1|   -1|       0| unknown| no|
| 58|  technician| married|  unknown|     no|     71|    yes|  no|unknown|  5|  may|      71|       1|   -1|       0| unknown| no|
| 57|    services| married|secondary|     no|    162|    yes|  no|unknown|  5|  may|     174|       1|   -1|       0| unknown| no|
| 51|     retired| married|  primary|     no|    229|    yes|  no|unknown|  5|  may|     353|       1|   -1|       0| unknown| no|
| 45|      admin.|  single|  unknown|     no|     13|    yes|  no|unknown|  5|  may|      98|       1|   -1|       0| unknown| no|
| 57| blue-collar| married|  primary|     no|     52|    yes|  no|unknown|  5|  may|      38|       1|   -1|       0| unknown| no|
| 60|     retired| married|  primary|     no|     60|    yes|  no|unknown|  5|  may|     219|       1|   -1|       0| unknown| no|
| 33|    services| married|secondary|     no|      0|    yes|  no|unknown|  5|  may|      54|       1|   -1|       0| unknown| no|
+---+------------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+---+
only showing top 20 rows
```

2. Give marketing success rate (No. of people subscribed / total no. of entries)

a. Give marketing failure rate

```
 scala> val total_count = df.count()

total_count: Long = 45211


scala>  val total_success_count = df.filter("y == 'yes'").count()

total_success_count: Long = 5289


scala> val total_failure_count = total - total_success_count

total_failure_count: Long = 39922


scala> val success_rate = (total_success_count*100)/total_count

success_rate: Long = 11
```

scala> val failure_rate = (total_failure_count*100)/total_count

failure_rate: Long = 88

```
scala> val total_count = df.count()
total_count: Long = 45211

scala>  val total_success_count = df.filter("y == 'yes'").count()
total_success_count: Long = 5289

scala> val total_failure_count = total - total_success_count
total_failure_count: Long = 39922

scala> val success_rate = (total_success_count*100)/total_count
success_rate: Long = 11

scala> val failure_rate = (total_failure_count*100)/total_count
failure_rate: Long = 88
```

3. Give the maximum, mean, and minimum age of the average targeted customer

scala> df.select("age").describe().show()

```
scala> df.select("age").describe().show()
+-------+------------------+
|summary|               age|
+-------+------------------+
|  count|             45211|
|   mean| 40.93621021432837|
| stddev|10.618762040975405|
|    min|                18|
|    max|                95|
+-------+------------------+
```

4. Check the quality of customers by checking average balance, median balance of customers

scala> df.agg(avg("balance")).show()

scala> df.select(median(col("balance"))).show()

```
scala> df.agg(avg("balance")).show()
+------------------+
|      avg(balance)|
+------------------+
|1362.2720576850766|
+------------------+


scala> df.select(median(col("balance"))).show()
+----------------+
|median(balance)|
+----------------+
|           448.0|
+----------------+
```

3. Check if age matters in marketing subscription for deposit

scala> val young_age_success_count = df.filter(df("age") < 31 && df("y") === "yes").count()

young_age_success_count: Long = 1145

scala> val mid_age_success_count = df.filter((df("age") >30 && df("age") < 46) && df("y") === "yes").count()

mid_age_success_count: Long = 2345

scala> val old_age_success_count = df.filter(df("age") > 45 && df("y") === "yes").count()

old_age_success_count: Long = 1799

scala> val young_age_success_rate = (young_age_success_count*100)/total_success_count

young_age_success_rate: Long = 21

scala> val mid_age_success_rate = (mid_age_success_count*100)/total_success_count

mid_age_success_rate: Long = 44

scala> val old_age_success_rate = (old_age_success_count*100)/total_success_count

old_age_success_rate: Long = 34

```
scala>  val young_age_success_count = df.filter(df("age") < 31 && df("y") === "yes").count()
young_age_success_count: Long = 1145

scala> val mid_age_success_count = df.filter((df("age") >30 && df("age") < 46) && df("y") === "yes").count()
mid_age_success_count: Long = 2345

scala>  val old_age_success_count = df.filter(df("age") > 45 && df("y") === "yes").count()
old_age_success_count: Long = 1799

scala> val young_age_success_rate = (young_age_success_count*100)/total_success_count
young_age_success_rate: Long = 21

scala> val mid_age_success_rate = (mid_age_success_count*100)/total_success_count
mid_age_success_rate: Long = 44

scala> val old_age_success_rate = (old_age_success_count*100)/total_success_count
old_age_success_rate: Long = 34
```

→ Based on the above success rate calculation of the various age groups, it is concluded that young-aged customers are showing the highest success rate while the middle-aged customers are showing the lowest success rate. Hence, we can say that the age matters in marketing subscription for deposit.

4. Check if marital status mattered for a subscription to deposit

scala> val single_success_count = df.filter(df("marital") === "single" && df("y") === "yes").count()

single_success_count: Long = 1912

scala> val married_success_count = df.filter(df("marital") === "married" && df("y") === "yes").count()

married_success_count: Long = 2755

scala> val divorced_success_count = df.filter(df("marital") === "divorced" && df("y") === "yes").count()

divorced_success_count: Long = 622

scala> val single_success_rate = (single_success_count*100)/total_success_count

single_success_rate: Long = 36

scala> val married_success_rate = (married_success_count*100)/total_success_count

married_success_rate: Long = 52

scala> val divorced_success_rate = (divorced_success_count*100)/total_success_count

divorced_success_rate: Long = 11

```
scala>  val single_success_count = df.filter(df("marital") === "single" && df("y") === "yes").count()
single_success_count: Long = 1912

scala>  val married_success_count = df.filter(df("marital") === "married" && df("y") === "yes").count()
married_success_count: Long = 2755

scala>  val divorced_success_count = df.filter(df("marital") === "divorced" && df("y") === "yes").count()
divorced_success_count: Long = 622

scala> val single_success_rate = (single_success_count*100)/total_success_count
single_success_rate: Long = 36

scala> val married_success_rate = (married_success_count*100)/total_success_count
married_success_rate: Long = 52

scala> val divorced_success_rate = (divorced_success_count*100)/total_success_count
divorced_success_rate: Long = 11
```

→ Based on the above success rate calculation of the various marital status categories, it is concluded that married customers are showing the highest success rate while the divorced customers are showing the lowest success rate. Hence, we can say that the marital status mattered in marketing subscription for deposit.

5. Check if age and marital status together mattered for a subscription to deposit scheme

total success count = 5289

Single total count = 1912

young = 945

middle = 839

old = 128

Married total count = 2755

young = 182

middle = 1247

old = 1326

Divorced total count = 622

young = 18

middle = 259

old = 345

```
scala> val single_young_age_success_count = df.filter(df("marital") === "single" && df("age") < 31 && df("y") === "yes").count()

single_young_age_success_count: Long = 945

scala> val single_mid_age_success_count = df.filter(df("marital") === "single" && (df("age") > 30 && df("age") < 46) && df("y") === "yes").count()

single_mid_age_success_count: Long = 839

scala> val single_old_age_success_count = df.filter(df("marital") === "single" && df("age") > 45 && df("y") === "yes").count()

single_old_age_success_count: Long = 128

scala> val single_young_age_success_rate = (single_young_age_success_count*100)/total_success_count

single_young_age_success_rate: Long = 17

scala> val single_mid_age_success_rate = (single_mid_age_success_count*100)/total_success_count

single_mid_age_success_rate: Long = 15

scala> val single_old_age_success_rate = (single_old_age_success_count*100)/total_success_count

single_old_age_success_rate: Long = 2

scala> val married_young_age_success_count = df.filter(df("marital") === "married" && df("age") < 31 && df("y") === "yes").count()

married_young_age_success_count: Long = 182

scala> val married_mid_age_success_count = df.filter(df("marital") === "married" && (df("age") > 30 && df("age") < 46) && df("y") === "yes").count()

married_mid_age_success_count: Long = 1247

scala> val married_old_age_success_count = df.filter(df("marital") === "married" && df("age") > 45 && df("y") === "yes").count()

married_old_age_success_count: Long = 1326
```

```
scala> val married_young_age_success_rate = (married_young_age_success_count*100)/total_success_count
married_young_age_success_rate: Long = 3

scala> val married_mid_age_success_rate = (married_mid_age_success_count*100)/total_success_count
married_mid_age_success_rate: Long = 23

scala> val married_old_age_success_rate = (married_old_age_success_count*100)/total_success_count
married_old_age_success_rate: Long = 25

scala> val divorced_young_age_success_count = df.filter(df("marital") === "divorced" && df("age") < 31 && df("y")
  === "yes").count()
divorced_young_age_success_count: Long = 18

scala> val divorced_mid_age_success_count = df.filter(df("marital") === "divorced" && (df("age") > 30 && df("age")
  < 46) && df("y") === "yes").count()
divorced_mid_age_success_count: Long = 259

scala>  val divorced_old_age_success_count = df.filter(df("marital") === "divorced" && df("age") > 45 && df("y") ===
  "yes").count()
divorced_old_age_success_count: Long = 345

scala> val divorced_young_age_success_rate = (divorced_young_age_success_count*100)/total_success_count
divorced_young_age_success_rate: Long = 0

scala> val divorced_mid_age_success_rate = (divorced_mid_age_success_count*100)/total_success_count
divorced_mid_age_success_rate: Long = 4

scala> val divorced_old_age_success_rate = (divorced_old_age_success_count*100)/total_success_count
divorced_old_age_success_rate: Long = 6
```

Category	Success Count	Success Rate (approx.)
Single-young	945	17%
Single-mid	839	15%
Single-old	128	2%
Married-young	182	3%
Married-mid	1247	23%

Married-old	1326	25%
Divorced-young	18	0.3%
Divorced-mid	259	4%
Divorced-old	345	6%
Total	**5289**	**100%**

```
scala> val single_young_age_success_count = df.filter(df("marital") === "single" && df("age") < 31 && df("y") === "yes").count()
single_young_age_success_count: Long = 945

scala> val single_mid_age_success_count = df.filter(df("marital") === "single" && (df("age") > 30 && df("age") < 46) && df("y") === "yes").count()
single_mid_age_success_count: Long = 839

scala> val single_old_age_success_count = df.filter(df("marital") === "single" && df("age") > 45 && df("y") === "yes").count()
single_old_age_success_count: Long = 128

scala> val single_young_age_success_rate = (single_young_age_success_count*100)/total_success_count
single_young_age_success_rate: Long = 17

scala> val single_mid_age_success_rate = (single_mid_age_success_count*100)/total_success_count
single_mid_age_success_rate: Long = 15

scala> val single_old_age_success_rate = (single_old_age_success_count*100)/total_success_count
single_old_age_success_rate: Long = 2

scala> val married_young_age_success_count = df.filter(df("marital") === "married" && df("age") < 31 && df("y") === "yes").count()
married_young_age_success_count: Long = 182

scala> val married_mid_age_success_count = df.filter(df("marital") === "married" && (df("age") > 30 && df("age") < 46) && df("y") === "yes").count()
married_mid_age_success_count: Long = 1247

scala> val married_old_age_success_count = df.filter(df("marital") === "married" && df("age") > 45 && df("y") === "yes").count()
married_old_age_success_count: Long = 1326

scala> val married_young_age_success_rate = (married_young_age_success_count*100)/total_success_count
married_young_age_success_rate: Long = 3

scala> val married_mid_age_success_rate = (married_mid_age_success_count*100)/total_success_count
married_mid_age_success_rate: Long = 23

scala> val married_old_age_success_rate = (married_old_age_success_count*100)/total_success_count
married_old_age_success_rate: Long = 25

scala> val divorced_young_age_success_count = df.filter(df("marital") === "divorced" && df("age") < 31 && df("y") === "yes").count()
divorced_young_age_success_count: Long = 18
```

```
scala> val divorced_mid_age_success_count = df.filter(df("marital") === "divorced" && (df("age") > 30 && df("age") < 46) && df("y") === "yes").count()
divorced_mid_age_success_count: Long = 259

scala>  val divorced_old_age_success_count = df.filter(df("marital") === "divorced" && df("age") > 45 && df("y") === "yes").count()
divorced_old_age_success_count: Long = 345

scala> val divorced_young_age_success_rate = (divorced_young_age_success_count*100)/total_success_count
divorced_young_age_success_rate: Long = 0

scala> val divorced_mid_age_success_rate = (divorced_mid_age_success_count*100)/total_success_count
divorced_mid_age_success_rate: Long = 4

scala> val divorced_old_age_success_rate = (divorced_old_age_success_count*100)/total_success_count
divorced_old_age_success_rate: Long = 6
```

→ Based on the above calculations and comparison, the below points are observed:

a. In the single category, young and middle-aged customers are showing the good success rate while the old age success rate is negligible.

b. In the married category, middle-aged and old-aged customers are showing the very good success rate while young category is very less.

c. In the divorced category, middle-aged and old-aged customers are showing good success rate while the young category is showing the negligible success.