**Q1)**

**Pseudocode :**

```
MAP(key, value)

    LS = split(value, '\t')
    event_epoch_time = LS[0]
    user_id = LS[1]
    device_id = LS[2]
    user_agent = LS[3]
    if ( event_epoch_time == NULL || user_id == NULL || device_id == NULL || user_agent == NULL)
            write("JUNK_RECORDS", value);
```

**Explanation :**

Assuming the "value" parameter is a tab separated elements, split "value" using delimiter /t and store the values in list variable LS
Index starts from 0 in the list LS, So LS[0] represents event_epoch_time value, LS[1] represents user_id, LS[2] represents device_id, and LS[3] represents user_agent.
"==" represents logical equals in pseudocode.
If any values of event_epoch_time, user_id, device_id, user_agent equals NULL, writing the corresponding records to the output with JUNK_RECORDS as a key.
Once executed, output dataset will have collection of junk records.

**Q2)**

**Pseudocode :**

```
MAP(key, value)

    LS = split(value, '\t')
    user_id = LS[1]
    user_agent = LS[3]
    user_agent_Info_LS = split(user_agent, ':')
    resultValue = user_agent_Info_LS[1] + '\t' + user_agent_Info_LS[0]
    write(user_id, resultValue)
```

**Explanation :**

Assuming the "value" parameter is a tab separated elements, split "value" using delimiter /t and store the values in list variable LS
Index starts from 0 in the list LS, So LS[0] represents event_epoch_time value, LS[1] represents user_id, LS[2] represents device_id, and LS[3] represents user_agent.
"==" represents logical equals in pseudocode.
Split user_agent_info using ':' delimiter, where user_agent_Info_LS[0] denotes platform and user_agent_Info_LS[1] denotes OS version.
Output user_id as key, and resultValue containing OSVersion and Platform separated by tab

Q3)

a. **Find out the number of veg and non-veg pizzas sold.**

**Pseudocode :**

```
MAP(key, value)
        LS = split(value, '\t')    // Assuming values are tab separated
        event_epoch_time = LS[0]
        user_id = LS[1]
        device_id = LS[2]
        user_agent = LS[3]
        isCheeseBurst = LS[5]
        isVeg = LS[11]
        if ( event_epoch_time != NULL &&user_id != NULL
                && device_id != NULL && user_agent != NULL)
                if(isVeg == 'Y')
                        getCounter("veg").incrementBy(1);
                else
                        getCounter("non_veg").incrementBy(1);


veg_pizzas_sold_count = write(getCounter("veg"))
nonveg_pizzas_sold_count = write(getCounter("non_veg"))
```

**Explanation:**

Mapper function takes key, value as an input. Where value is a record with tab separated values. Split value with delimiter '\t' and skip the junk records.
When isVeg value is 'Y' i.e., if it is veg, use counter variable "veg"(Create if doesn't exist) and increment counter variable by 1.
When isVeg value is 'N' i.e., if it is non-veg, use counter variable "non_veg"(Create if doesn't exist) and increment counter variable by 1.
After the mapper function,
To find veg pizzas sold is to get the value of counter "veg" i.e., returns 3
To find non veg pizzas sold is to get the value of counter "non_veg" i.e., return 1

b. **Find out the size wise distribution of pizzas sold.**

**Pseudocode :**

```
MAP(key, value)
        LS = split(value, '\t')    // Assuming values are tab separated
        event_epoch_time = LS[0]
        user_id = LS[1]
        device_id = LS[2]
        user_agent = LS[3]
        isCheeseBurst = LS[5]
        isSize = LS[6]
```

```
        if ( event_epoch_time != NULL &&user_id != NULL
                && device_id != NULL && user_agent != NULL)
                getCounter(isSize).incrementBy(1);

regular_size_pizzas_sold_count = write(getCounter("R"))
medium_size_pizzas_sold_count = write(getCounter("M"))
large_size_pizzas_sold_count = write(getCounter("L"))
```

**Explanation:**

Mapper function takes key, value as an input. Where value is a record with tab separated values.
Split value with delimiter '\t' and skip the junk records.
Use counter variable for sizes(M,R,L) (Create if doesn't exist) and increment counter variable by 1.
After the mapper function,
To find pizzas based on sizes , we just need to get the value of counter variables respectively i.e.,
For regular, write(getCounter("R")) which returns 1
For Medium, write(getCounter("M")) which returns 2
For Large, write(getCounter("L")) which returns 1

c. **Find out how many cheese burst pizzas were sold.**

**Pseudocode :**

```
MAP(key, value)
        LS = split(value, '\t')    // Assuming values are tab separated
        event_epoch_time = LS[0]
        user_id = LS[1]
        device_id = LS[2]
        user_agent = LS[3]
        isCheeseBurst = LS[5]
        if ( event_epoch_time != NULL &&user_id != NULL
                && device_id != NULL && user_agent != NULL)
                if(isCheeseBurst == 'Y')
                        getCounter("cheeseBurstCount").incrementBy(1);


cheeseburst_pizzas_sold_count = write(getCounter("cheeseBurstCount"))
```

**Explanation:**

Mapper function takes key, value as an input. Where value is a record with tab separated values.
Split value with delimiter '\t' and skip the junk records.
Use counter variable cheeseBurstCount (Create if doesn't exist) and increment counter variable by 1.
After the mapper function,
To find count of cheese burst pizzas sold , we just need to get the value of counter variables respectively i.e., write(getCounter("cheeseBurstCount")) which returns 3

**d. Find out how many regular cheese burst pizzas were sold.**

**Pseudocode :**

```
MAP(key, value)
        LS = split(value, '\t')    // Assuming values are tab separated
        event_epoch_time = LS[0]
        user_id = LS[1]
        device_id = LS[2]
        user_agent = LS[3]
        isCheeseBurst = LS[5]
        size = LS[6]
        if ( event_epoch_time != NULL &&user_id != NULL
                && device_id != NULL && user_agent != NULL)
                if(isCheeseBurst == 'Y' && size == 'R' )
                        getCounter("regularCheeseBurstCount").incrementBy(1);


regular _cheeseburst_pizzas_sold_count = write(getCounter("regularCheeseBurstCount"))
```

**Explanation:**

Mapper function takes key, value as an input. Where value is a record with tab separated values. Split value with delimiter '\t' and skip the junk records.

Use counter variable regularCheeseBurstCount (Create if doesn't exist) and increment counter variable by 1.

After the mapper function,

To find count of regular cheese burst pizzas sold , we just need to get the value of counter variable respectively i.e., write(getCounter("regularCheeseBurstCount")) which returns 0

**e. Find out the number of cheese burst pizzas whose cost is below Rs 500.**

**Pseudocode :**

```
MAP(key, value)
        LS = split(value, '\t')    // Assuming values are tab separated
        event_epoch_time = LS[0]
        user_id = LS[1]
        device_id = LS[2]
        user_agent = LS[3]
        isCheeseBurst = LS[5]
        price = ConvertToInt(LS[8])   // Convert string to integer value
        if ( event_epoch_time != NULL &&user_id != NULL
                && device_id != NULL && user_agent != NULL)
                if(isCheeseBurst == 'Y' && price < 500)
                        getCounter("cheeseBurst_PriceRange_500_Count").incrementBy(1);
```

count = write(getCounter("cheeseBurst_PriceRange_500_Count"))

**Explanation:**
Mapper function takes key, value as an input. Where value is a record with tab separated values. Split value with delimiter '\t' and skip the junk records.
Use counter variable cheeseBurst_PriceRange_500_Count (Create if doesn't exist) and increment counter variable by 1.
After the mapper function,
To find count of regular cheese burst pizzas sold , we just need to get the value of counter variable respectively i.e., write(getCounter("cheeseBurst_PriceRange_500_Count")) which returns 1

**Q4)**

1. **Find out the number of veg and non veg pizzas sold.**
   **Pseudocode :**

```
MAP(key, value)
        LS = split(value, '\t')    // Assuming values are tab separated
        event_epoch_time = LS[0]
        user_id = LS[1]
        device_id = LS[2]
        user_agent = LS[3]
        isVeg = LS[11]
        if ( event_epoch_time != NULL && user_id != NULL && device_id != NULL
                && user_agent != NULL)
                write(isVeg, 1);
Reduce(key, valueList)
        count = 0
        for i = 1 to valueList.Length
                count = count + 1;
        write(key, count)
```

**Explanation :**
Mapper function reads the input dataset , skips the junk records and writes output with keys (Y or N) and value 1.
At the end of mapper functions, output will be aggregated so that there are two sets with keys Y and N and its associated list of values. i.e., ex:- (Y, {1,1,1}) and (N, {1})
Reduce function takes key and valuelist as an input, and returns the count of values which denotes number of pizzas sold associated with that key.
Therefore
Count of Veg pizzas sold is calculated with Y as key. Result is (Y, 3)
Count of Non-Veg pizzas sold is calculated with N as key. Result is (N, 1)

2. **Find out the size wise distribution of pizzas sold.**

**Pseudocode :**

```
MAP(key, value)

        LS = split(value, '\t')    // Assuming values are tab separated
        event_epoch_time = LS[0]
        user_id = LS[1]
        device_id = LS[2]
        user_agent = LS[3]
        size = LS[6]
        if ( event_epoch_time != NULL && user_id != NULL && device_id != NULL
                &&  user_agent != NULL)
                write(size, 1);

Reduce(key, valueList)
        count = 0
        for i = 1 to valueList.Length
                count = count + 1;
        write(key, count)
```

**Explanation :**

Mapper function reads the input dataset , skips the junk records and writes output with keys (M, R, L) and value 1.

At the end of mapper functions, output will be aggregated so that there are two sets with keys Y and N and its associated list of values. i.e., ex:- (M, {1,1}), (R, {1}) (L, {1})

Reduce function takes key and valuelist as an input, and returns the count of values which denotes number of pizzas sold associated with that key.

3. **Find out how many cheese burst pizzas were sold.**

**Pseudocode :**

```
MAP(key, value)

        LS = split(value, '\t')    // Assuming values are tab separated
        event_epoch_time = LS[0]
        user_id = LS[1]
        device_id = LS[2]
        user_agent = LS[3]
        isCheeseBurst = LS[5]
        if ( event_epoch_time != NULL && user_id != NULL && device_id != NULL
                &&  user_agent != NULL)
                if (isCheeseBurst == 'Y')
                        write(isCheeseBurst, 1);
Reduce(key, valueList)
        count = 0
        for i = 1 to valueList.Length
```

```
        count = count + 1;
    write(key, count)
```

**Explanation :**

Mapper function reads the input dataset , skips the junk records and writes output with key (Y) and value 1.

At the end of mapper functions, output will be aggregated so that there will be only one set with key Y and its associated list of values. i.e., ex:- (Y, {1,1,1})

Reduce function takes key and valuelist as an input, and returns the count of values which denotes number of pizzas sold associated with that key.

Therefore

Count of Cheese burst pizzas sold is calculated with Y as key. Result is (Y, 3)

4. **Find out how many regular cheese burst pizzas were sold.**

**Pseudocode :**

```
MAP(key, value)
    LS = split(value, '\t')    // Assuming values are tab separated
    event_epoch_time = LS[0]
    user_id = LS[1]
    device_id = LS[2]
    user_agent = LS[3]
    isCheeseBurst = LS[5]
    size = LS[6]
    if ( event_epoch_time != NULL && user_id != NULL && device_id != NULL
        &&  user_agent != NULL)
        If(isCheeseBurst == 'Y' && size == 'R')
            write("RegularCheeseBurst", 1);
Reduce(key, valueList)
    count = 0
    for i = 1 to valueList.Length
        count = count + 1;
    write(key, count)
```

**Explanation :**

Mapper function reads the input dataset , skips the junk records and writes output with only one key RegularCheeseBurst and value 1.

At the end of mapper functions, output will be aggregated so that there are only one set with key RegularCheeseBurst and its associated list of values. i.e., ex:- (RegularCheeseBurst, {})

Reduce function takes key and valuelist as an input, and returns the count of values which denotes number of pizzas sold associated with that key.

Therefore

Count of regular cheese burst pizzas sold is calculated with RegularCheeseBurst as key. Result is (RegularCheeseBurst, 0)

5. **Find out the number of cheese burst pizzas whose cost is below Rs 500.**

**Pseudocode :**

```
MAP(key, value)
        LS = split(value, '\t')    // Assuming values are tab separated
        event_epoch_time = LS[0]
        user_id = LS[1]
        device_id = LS[2]
        user_agent = LS[3]
        isCheeseBurst = LS[5]
        size = LS[6]
        price = ConvertToInt(LS[8]) // Convert string to integer
        if ( event_epoch_time != NULL && user_id != NULL && device_id != NULL
                &&  user_agent != NULL)
                if(isCheeseBurst == 'Y' && price < 500)
                        write("cheeseBurst_priceRange_500", 1);
Reduce(key, valueList)
        count = 0
        for i = 1 to valueList.Length
                count = count + 1;
        write(key, count)
```

**Explanation :**

Mapper function reads the input dataset , skips the junk records and writes output with key "cheeseBurst_priceRange_500" and value 1.

At the end of mapper functions, output will be aggregated so that there will be only one set with key "cheeseBurst_priceRange_500" and its associated list of values. i.e., ex:- (cheeseBurst_priceRange_500, {1})

Reduce function takes key and valuelist as an input, and returns the count of values which denotes number of pizzas sold associated with that key.

Therefore

Count of cheese burst with price less than 500 pizzas sold is calculated with cheeseBurst_priceRange_500 as key. Result is (cheeseBurst_priceRange_500, 1)