PySpark Code to Calculate PSI

```
from pyspark.sql import functions as F
from pyspark.sql.window import Window

# Step 1: Define baseline and validation datasets
baseline_df = spark_df.filter(F.col("Period") == "2023-01-01")
validation_df = spark_df.filter(F.col("Period") == "2023-02-01")

# Step 2: Apply binning to the baseline dataset
PSI_N_BINS = 10
ESTIMATED_PROBABILITY_FIELD = "Estimated_probability_Of_Default"

# Add binning using NTILE
window_spec = Window.orderBy(F.col(ESTIMATED_PROBABILITY_FIELD))
baseline_binned = baseline_df.withColumn(
   "bin", F.ntile(PSI_N_BINS).over(window_spec)
)

# Step 3: Summarize baseline bins
baseline_summary = (
   baseline_binned.groupBy("bin")
   .agg(
      F.min(ESTIMATED_PROBABILITY_FIELD).alias("MIN_PROB_1"),
      F.max(ESTIMATED_PROBABILITY_FIELD).alias("MAX_PROB_1"),
      F.count("*").alias("n_baseline"),
   )
)

# Step 4: Assign bins to the validation dataset
validation_with_bins = validation_df.join(
   baseline_summary,
```

```
    (F.col(ESTIMATED_PROBABILITY_FIELD) >= F.col("MIN_PROB_1"))
    & (F.col(ESTIMATED_PROBABILITY_FIELD) <= F.col("MAX_PROB_1")),
    "left",
).groupBy("bin").agg(F.count("*").alias("n_validation"))

# Step 5: Calculate PSI metrics
psi_table = (
    baseline_summary.join(validation_with_bins, "bin", "outer")
    .fillna(0, subset=["n_baseline", "n_validation"])
    .withColumn("p_baseline", F.col("n_baseline") /
F.sum("n_baseline").over(Window.partitionBy()))
    .withColumn("p_validation", F.col("n_validation") /
F.sum("n_validation").over(Window.partitionBy()))
    .withColumn("Difference", F.col("p_baseline") - F.col("p_validation"))
    .withColumn("Ratio", F.when(F.col("p_validation") != 0, F.col("p_baseline") /
F.col("p_validation")).otherwise(0))
    .withColumn("Weight_of_Evidence", F.when(F.col("Ratio") > 0,
F.log(F.col("Ratio"))).otherwise(0))
    .withColumn("Contribution", F.col("Difference") * F.col("Weight_of_Evidence"))
)

# Step 6: Calculate the total PSI
total_psi = psi_table.agg(F.sum("Contribution").alias("PSI")).collect()[0]["PSI"]

# Step 7: Display the results
print("PSI:", total_psi)
psi_table.show()
```

PySpark Code for VDI/CSI (Categorical Variables)

```
# Step 1: Define baseline and validation datasets
baseline_df = spark_df.filter(F.col("Period") == "2023-01-01")
validation_df = spark_df.filter(F.col("Period") == "2023-02-01")

# Step 2: Group by the categorical variable and calculate counts
categorical_field = "Category_Variable"

baseline_summary = (
   baseline_df.groupBy(categorical_field)
   .agg(F.count("*").alias("n_baseline"))
   .withColumn("p_baseline", F.col("n_baseline") /
F.sum("n_baseline").over(Window.partitionBy()))
)

validation_summary = (
   validation_df.groupBy(categorical_field)
   .agg(F.count("*").alias("n_validation"))
   .withColumn("p_validation", F.col("n_validation") /
F.sum("n_validation").over(Window.partitionBy()))
)

# Step 3: Join baseline and validation summaries
vdi_table = (
   baseline_summary.join(validation_summary, categorical_field, "outer")
   .fillna(0, subset=["n_baseline", "n_validation"])
   .withColumn("Difference", F.col("p_baseline") - F.col("p_validation"))
   .withColumn("Ratio", F.when(F.col("p_validation") != 0, F.col("p_baseline") /
F.col("p_validation")).otherwise(0))
   .withColumn("Weight_of_Evidence", F.when(F.col("Ratio") > 0,
F.log(F.col("Ratio"))).otherwise(0))
```

```
    .withColumn("Contribution", F.col("Difference") * F.col("Weight_of_Evidence"))
)

# Step 4: Calculate total VDI/CSI
total_vdi = vdi_table.agg(F.sum("Contribution").alias("VDI")).collect()[0]["VDI"]

# Step 5: Display the results
print("VDI (Categorical):", total_vdi)
vdi_table.show()
```

PySpark Code for Multiple Categorical Variables

```
# Step 1: Define the categorical variables and baseline/validation datasets
categorical_variables = ["Category_Variable_1", "Category_Variable_2", "Category_Variable_3"]
baseline_df = spark_df.filter(F.col("Period") == "2023-01-01")
validation_df = spark_df.filter(F.col("Period") == "2023-02-01")

# Step 2: Define a function to calculate VDI for a single categorical variable
def calculate_vdi_for_category(variable):
    baseline_summary = (
        baseline_df.groupBy(variable)
        .agg(F.count("*").alias("n_baseline"))
        .withColumn("p_baseline", F.col("n_baseline") /
F.sum("n_baseline").over(Window.partitionBy()))
    )

    validation_summary = (
        validation_df.groupBy(variable)
        .agg(F.count("*").alias("n_validation"))
        .withColumn("p_validation", F.col("n_validation") /
F.sum("n_validation").over(Window.partitionBy()))
    )
```

```
    vdi_table = (
        baseline_summary.join(validation_summary, variable, "outer")
        .fillna(0, subset=["n_baseline", "n_validation"])
        .withColumn("Difference", F.col("p_baseline") - F.col("p_validation"))
        .withColumn("Ratio", F.when(F.col("p_validation") != 0, F.col("p_baseline") / F.col("p_validation")).otherwise(0))
        .withColumn("Weight_of_Evidence", F.when(F.col("Ratio") > 0, F.log(F.col("Ratio"))).otherwise(0))
        .withColumn("Contribution", F.col("Difference") * F.col("Weight_of_Evidence"))
    )

    total_vdi = vdi_table.agg(F.sum("Contribution").alias("VDI")).collect()[0]["VDI"]
    return total_vdi, vdi_table

# Step 3: Loop through categorical variables and calculate VDI for each
vdi_results = {}
for variable in categorical_variables:
    vdi_value, vdi_table = calculate_vdi_for_category(variable)
    vdi_results[variable] = {"VDI": vdi_value, "Table": vdi_table}

# Step 4: Display results for all variables
for variable, result in vdi_results.items():
    print(f"VDI for {variable}: {result['VDI']}")
    print(f"Table for {variable}:")
    result["Table"].show()
```

PySpark Code for VDI/CSI (Continuous Variables)

```
# Step 1: Define baseline and validation datasets
baseline_df = spark_df.filter(F.col("Period") == "2023-01-01")
validation_df = spark_df.filter(F.col("Period") == "2023-02-01")
```

```
# Step 2: Define the continuous variable and number of bins
continuous_field = "Continuous_Variable"
VDI_N_BINS = 10

# Step 3: Bin the baseline dataset using NTILE
window_spec = Window.orderBy(F.col(continuous_field))
baseline_binned = baseline_df.withColumn(
   "bin", F.ntile(VDI_N_BINS).over(window_spec)
)

# Step 4: Summarize baseline bins
baseline_summary = (
   baseline_binned.groupBy("bin")
   .agg(
      F.min(continuous_field).alias("MIN_VALUE"),
      F.max(continuous_field).alias("MAX_VALUE"),
      F.count("*").alias("n_baseline"),
   )
)

# Step 5: Assign bins to the validation dataset
validation_with_bins = validation_df.join(
   baseline_summary,
   (F.col(continuous_field) >= F.col("MIN_VALUE"))
   & (F.col(continuous_field) <= F.col("MAX_VALUE")),
   "left",
).groupBy("bin").agg(F.count("*").alias("n_validation"))

# Step 6: Calculate VDI metrics
vdi_table = (
   baseline_summary.join(validation_with_bins, "bin", "outer")
```

```
    .fillna(0, subset=["n_baseline", "n_validation"])
    .withColumn("p_baseline", F.col("n_baseline") / F.sum("n_baseline").over(Window.partitionBy()))
    .withColumn("p_validation", F.col("n_validation") / F.sum("n_validation").over(Window.partitionBy()))
    .withColumn("Difference", F.col("p_baseline") - F.col("p_validation"))
    .withColumn("Ratio", F.when(F.col("p_validation") != 0, F.col("p_baseline") / F.col("p_validation")).otherwise(0))
    .withColumn("Weight_of_Evidence", F.when(F.col("Ratio") > 0, F.log(F.col("Ratio"))).otherwise(0))
    .withColumn("Contribution", F.col("Difference") * F.col("Weight_of_Evidence"))
)

# Step 7: Calculate total VDI/CSI
total_vdi = vdi_table.agg(F.sum("Contribution").alias("VDI")).collect()[0]["VDI"]

# Step 8: Display the results
print("VDI (Continuous):", total_vdi)
vdi_table.show()
```

PySpark Code for Applying VDI/CSI to Multiple Continuous Variables

```
# Step 1: Define the continuous variables and baseline/validation datasets
continuous_variables = ["Continuous_Variable_1", "Continuous_Variable_2", "Continuous_Variable_3"]
baseline_df = spark_df.filter(F.col("Period") == "2023-01-01")
validation_df = spark_df.filter(F.col("Period") == "2023-02-01")
VDI_N_BINS = 10  # Number of bins for all continuous variables

# Step 2: Define a function to calculate VDI for a single continuous variable
def calculate_vdi_for_continuous(variable):
    # Bin the baseline dataset
    window_spec = Window.orderBy(F.col(variable))
```

```
  baseline_binned = baseline_df.withColumn("bin", F.ntile(VDI_N_BINS).over(window_spec))

  # Summarize baseline bins
  baseline_summary = (
    baseline_binned.groupBy("bin")
    .agg(
      F.min(variable).alias("MIN_VALUE"),
      F.max(variable).alias("MAX_VALUE"),
      F.count("*").alias("n_baseline"),
    )
  )

  # Assign bins to the validation dataset
  validation_with_bins = validation_df.join(
    baseline_summary,
    (F.col(variable) >= F.col("MIN_VALUE"))
    & (F.col(variable) <= F.col("MAX_VALUE")),
    "left",
  ).groupBy("bin").agg(F.count("*").alias("n_validation"))

  # Calculate VDI metrics
  vdi_table = (
    baseline_summary.join(validation_with_bins, "bin", "outer")
    .fillna(0, subset=["n_baseline", "n_validation"])
    .withColumn("p_baseline", F.col("n_baseline") /
F.sum("n_baseline").over(Window.partitionBy()))
    .withColumn("p_validation", F.col("n_validation") /
F.sum("n_validation").over(Window.partitionBy()))
    .withColumn("Difference", F.col("p_baseline") - F.col("p_validation"))
    .withColumn("Ratio", F.when(F.col("p_validation") != 0, F.col("p_baseline") /
F.col("p_validation")).otherwise(0))
```

```
        .withColumn("Weight_of_Evidence", F.when(F.col("Ratio") > 0,
F.log(F.col("Ratio"))).otherwise(0))
        .withColumn("Contribution", F.col("Difference") * F.col("Weight_of_Evidence"))
    )

    # Calculate total VDI
    total_vdi = vdi_table.agg(F.sum("Contribution").alias("VDI")).collect()[0]["VDI"]
    return total_vdi, vdi_table

# Step 3: Loop through continuous variables and calculate VDI for each
vdi_results = {}
for variable in continuous_variables:
    vdi_value, vdi_table = calculate_vdi_for_continuous(variable)
    vdi_results[variable] = {"VDI": vdi_value, "Table": vdi_table}

# Step 4: Display results for all variables
for variable, result in vdi_results.items():
    print(f"VDI for {variable}: {result['VDI']}")
    print(f"Table for {variable}:")
    result["Table"].show()
```