Data Transformation Class

```python
## Author: Goh Boon Xiang
from pyspark.ml.feature import StringIndexer, OneHotEncoder
from pyspark.sql import DataFrame, SparkSession
from pyspark.sql.functions import udf, col
from pyspark.sql.types import ArrayType, StringType
from pyspark.ml.feature import Tokenizer, HashingTF, IDF
import pyspark.sql.functions as F
from nltk.stem import WordNetLemmatizer

class DataTransformations:

    @staticmethod
    def tokenize(df: DataFrame, input_col: str, output_col: str) -> DataFrame:
        tokenizer = Tokenizer(inputCol=input_col, outputCol=output_col)
        return tokenizer.transform(df)

    @staticmethod
    def lemmatize_tokens(df: DataFrame, tokens_col: str) -> DataFrame:
        lemmatizer = WordNetLemmatizer()

        def lemmatize(tokens):
            return [lemmatizer.lemmatize(token) for token in tokens]

        lemmatize_udf = udf(lemmatize, ArrayType(StringType()))
        return df.withColumn(tokens_col, lemmatize_udf(col(tokens_col)))
```

```
    @staticmethod
    def calculate_tfidf(df: DataFrame, tokens_col: str, raw_features_col: str = "rawFeatures", features_col: str = "features", num_features: int = 10000) -> DataFrame:
        hashingTF = HashingTF(inputCol=tokens_col, outputCol=raw_features_col, numFeatures=num_features)
        featurized_df = hashingTF.transform(df)

        idf = IDF(inputCol=raw_features_col, outputCol=features_col)
        idf_model = idf.fit(featurized_df)
        return idf_model.transform(featurized_df)

    @staticmethod
    def oversample(df: DataFrame, label_col: str, majority_label: int, minority_label: int) -> DataFrame:
        major_df = df.filter(col(label_col) == majority_label)
        minor_df = df.filter(col(label_col) == minority_label)

        major_count = major_df.count()
        minor_count = minor_df.count()

        if minor_count > 0:
            ratio = int(major_count / minor_count)
            if ratio > 1:
                oversampled_df = minor_df.withColumn("dummy", F.explode(F.array([F.lit(x) for x in range(ratio)]))).drop('dummy')
                return major_df.union(oversampled_df)
            else:
                return df
        else:
            return df
```

```
    @staticmethod
    def one_hot_encode(df: DataFrame, categorical_col: str) -> DataFrame:
        """Perform one-hot encoding on the specified categorical column."""
        # Index the categorical column
        indexer = StringIndexer(inputCol=categorical_col, outputCol=f"{categorical_col}_index")
        indexed_df = indexer.fit(df).transform(df)

        # One-Hot Encode the indexed column
        encoder = OneHotEncoder(inputCols=[f"{categorical_col}_index"], outputCols=[f"{categorical_col}_encoded"])
        encoded_df = encoder.fit(indexed_df).transform(indexed_df)

        return encoded_df
```