# Introduction to PySpark (Machine Learning Pipline)

## String to Integer

Now you'll use the `.cast()` method you learned in the previous exercise to convert all the appropriate columns from your DataFrame `model_data` to integers!

To convert the type of a column using the `.cast()` method, you can write code like this:

```
dataframe = dataframe.withColumn("col", dataframe.col.cast("new_type"))
```

```
# Cast the columns to integers
model_data = model_data.withColumn("arr_delay", model_data.arr_delay.cast("integer"))
model_data = model_data.withColumn("air_time", model_data.air_time.cast("integer"))
model_data = model_data.withColumn("month", model_data.month.cast('integer'))
model_data = model_data.withColumn("plane_year", model_data.plane_year.cast('integer'))
```

## Create New Column

In the last exercise, you converted the column `plane_year` to an integer. This column holds the year each plane was manufactured. However, your model will use the planes' *age*, which is slightly different from the year it was made!

```
# Create the column plane_age
model_data = model_data.withColumn("plane_age", model_data.year - model_data.plane_year)
```

## Making a Boolean

Consider that you're modeling a yes or no question: is the flight late? However, your data contains the arrival delay in minutes for each flight. Thus, you'll need to create a boolean column which indicates whether the flight was late or not!

```
# Create is_late
model_data = model_data.withColumn("is_late", model_data.arr_delay > 0)

```

```
# Convert to an integer
model_data = model_data.withColumn("label", model_data.is_late.cast("integer"))

# Remove missing values
model_data = model_data.filter("arr_delay is not NULL and dep_delay is not NULL and air_time is not NULL and plane_year is not NULL")
```

## Carrier

In this exercise you'll create a `StringIndexer` and a `OneHotEncoder` to code the `carrier` column. To do this, you'll call the class constructors with the arguments `inputCol` and `outputCol`.

The `inputCol` is the name of the column you want to index or encode, and the `outputCol` is the name of the new column that the `Transformer` should create.

```
# Create a StringIndexer
carr_indexer = StringIndexer(inputCol="carrier", outputCol="carrier_index")

# Create a OneHotEncoder
carr_encoder = OneHotEncoder(inputCol="carrier_index", outputCol="carrier_fact")

# Create a StringIndexer
dest_indexer = StringIndexer(inputCol="dest", outputCol="dest_index")

# Create a OneHotEncoder
dest_encoder = OneHotEncoder(inputCol="dest_index", outputCol="dest_fact")
```

## Assemble a Vector

The last step in the `Pipeline` is to combine all of the columns containing our features into a single column. This has to be done before modeling can take place because every Spark modeling routine expects the data to be in this form. You can do this by storing each of the values from a column as an entry in a vector. Then, from the model's point of view, every observation is a vector that contains all of the information about it and a label that tells the modeler what value that observation corresponds to.

Because of this, the `pyspark.ml.feature` submodule contains a class called `VectorAssembler`. This `Transformer` takes all of the columns you specify and combines them into a new vector column.

```
# Make a VectorAssembler
vec_assembler = VectorAssembler(inputCols=["month", "air_time", "carrier_fact", "dest_fact", "plane_age"], outputCol="features")
```

## Create the Pipeline

You're finally ready to create a `Pipeline` !

`Pipeline` is a class in the `pyspark.ml` module that combines all the `Estimators` and `Transformers` that you've already created. This lets you reuse the same modeling process over and over again by wrapping it up in one simple object. Neat, right?

```
# Import Pipeline
from pyspark.ml import Pipeline

# Make the pipeline
flights_pipe = Pipeline(stages=[dest_indexer, dest_encoder, carr_indexer, carr_encoder,
vec_assembler])
```

## Transform the data

Hooray, now you're finally ready to pass your data through the `Pipeline` you created!

```
# Fit and transform the data
piped_data = flights_pipe.fit(model_data).transform(model_data)
```

## Split the Data

Now that you've done all your manipulations, the last step before modeling is to split the data!

```
# Split the data into training and test sets
training, test = piped_data.randomSplit([0.6, 0.4])
```