Pricing Validation and Auto-Correction

Objective

This notebook implements a Python function that validates and automatically corrects motor insurance product prices according to a set of business rules related to:

- product level
- coverage variant
- deductible amount

The final output includes:

1. Dictionary with corrected prices
2. Report of detected pricing inconsistencies.

In [9]:
```python
# I import basic typing tools.

from typing import Dict, Tuple, Optional, List
```

Business rules and reference values

In [10]:
```python
# I am going to create the different dictionaries that I need to use in the functio

# Here, I create a varibale that I am going to use as a dictionary where the key wi
# a text and the value a decimal number.

PriceTable = Dict[str, float]

# Here, I create a dictionary for the different products.

BASE_PRODUCT_PRICE: Dict[str, float] = {
    "mtpl": 400.0,
    "limited_casco": 700.0,
    "casco": 900.0,
}

# This dictionary will be one for the different types of variants.
# I give to compact and basic the same value and I treat them as the base line
# and I increase for comfort and premium based on the PDF.

VARIANT_FACTOR: Dict[str, float] = {
    "compact": 1.00,
    "basic": 1.00,
    "comfort": 1.07,
    "premium": 1.14,
}
```

```
# This one is for the different types of deductibles.
# The base one is 100, which I consider as a 100% and then the rest are based on
# the explanation provided in the PDF.

DEDUCTIBLE_FACTOR: Dict[int, float] = {
    100: 1.00,
    200: 0.90,
    500: 0.80,
}
```

Dividing products into components

In [11]:
```
# I am going to create a function which is going to divide the insurance product in
# parts. The first one will be the "main part", let's say the generic part and the
# will be the variant and the deductible. I will store them in a tuple that will be
# output of the function that I will use later on to calculate the prices.


def parse_price_key(key: str) -> Tuple[str, Optional[str], Optional[int]]:

    if key == "mtpl":
        return "mtpl", None, None

    if key.startswith("limited_casco_"):
        parts = key.split("_")
        product = "limited_casco"
        variant = parts[2]
        deductible = int(parts[3])
        return product, variant, deductible

    if key.startswith("casco_"):
        parts = key.split("_")
        product = "casco"
        variant = parts[1]
        deductible = int(parts[2])
        return product, variant, deductible

    raise ValueError(f"Unknown key: {key}")
```

Expected price calculation

In [12]:
```
# This is the function which will calculate the price for each product based on its
# general cost plus the variant and the deductible.
# First, I apply the variant to the base price because base on the guidelines it sh
# like this and then the deductible, also it makes sense logically. ALthough,
# mathematically it wouldn't change anything if I apply first the deductible and
# then the variant.

def expected_price(product: str, variant: Optional[str], deductible: Optional[int])

    price = BASE_PRODUCT_PRICE[product]

    if variant is not None:
```

```
        price *= VARIANT_FACTOR[variant]

    if deductible is not None:
        price *= DEDUCTIBLE_FACTOR[deductible]

# Here I round the price with 2 decimals because the prices to compare with have 2
# decimals also.

    return round(price, 2)
```

Validation and auto-correction

In [13]:
```
# This function will have a dictionary as input and a tuple as output which contain
# a dictionary and a list.
# First of all, I create a dictionaty for the corrected prices and a list called "m
# which I will use for those products that whose original price and expected price
# matching each other.
# After this, I will calculate the expected costs for every product with the two fu
# created before for later on, compare the original with the expected cost for each
# and as a mentioned, for each product that the prices don't match between them
# it will be stored in a list with a message.
# The output will be a dictionary with the product and the calculated cost and a li
# with a message for each product whose original and expected products don't match

def validate_and_correct_prices(prices: PriceTable) -> Tuple[PriceTable, List[str]]

    corrected: PriceTable = {}
    messages: List[str] = []

    for key, original_price in prices.items():
        product, variant, deductible = parse_price_key(key)
        expected = expected_price(product, variant, deductible)
        corrected[key] = expected

        if round(original_price, 2) != expected:
            message = f"{key}: {original_price} -> {expected}"
            messages.append(message)

    return corrected, messages
```

Example input data

In [14]:
```
# Data from the PDF attached.

example_prices: PriceTable = {
    "mtpl": 400,
    "limited_casco_compact_100": 820,
    "limited_casco_compact_200": 760,
    "limited_casco_compact_500": 650,
    "limited_casco_basic_100": 900,
    "limited_casco_basic_200": 780,
    "limited_casco_basic_500": 600,
    "limited_casco_comfort_100": 950,
```

```
    "limited_casco_comfort_200": 870,
    "limited_casco_comfort_500": 720,
    "limited_casco_premium_100": 1100,
    "limited_casco_premium_200": 980,
    "limited_casco_premium_500": 800,
    "casco_compact_100": 750,
    "casco_compact_200": 700,
    "casco_compact_500": 620,
    "casco_basic_100": 830,
    "casco_basic_200": 760,
    "casco_basic_500": 650,
    "casco_comfort_100": 900,
    "casco_comfort_200": 820,
    "casco_comfort_500": 720,
    "casco_premium_100": 1050,
    "casco_premium_200": 950,
    "casco_premium_500": 780,
}
```

Running the correction

In [15]:

```
# I am going to execute the code here.
# I call the function that I create for validation and I will store the output in t
# Corrected_prices and log (It will contain the messages for these products whose p
# don't match).
# After, it will print each product with the fixed price.
# Then, it will print the message for the products whose original and expected pric
# don't match or if there is none, it will print "No inconsistencies"

if __name__ == "__main__":
    corrected_prices, log = validate_and_correct_prices(example_prices)

    print("FIXED PRICES")
    for key in sorted(corrected_prices):
        print(key, "->", corrected_prices[key])

    print("DETECTED INCONSISTENCIES")
    if not log:
        print("There is no inconsistencies")
    else:
        for i in log:
            print("- ", i)
```

```
FIXED PRICES
casco_basic_100 -> 900.0
casco_basic_200 -> 810.0
casco_basic_500 -> 720.0
casco_comfort_100 -> 963.0
casco_comfort_200 -> 866.7
casco_comfort_500 -> 770.4
casco_compact_100 -> 900.0
casco_compact_200 -> 810.0
casco_compact_500 -> 720.0
casco_premium_100 -> 1026.0
casco_premium_200 -> 923.4
casco_premium_500 -> 820.8
limited_casco_basic_100 -> 700.0
limited_casco_basic_200 -> 630.0
limited_casco_basic_500 -> 560.0
limited_casco_comfort_100 -> 749.0
limited_casco_comfort_200 -> 674.1
limited_casco_comfort_500 -> 599.2
limited_casco_compact_100 -> 700.0
limited_casco_compact_200 -> 630.0
limited_casco_compact_500 -> 560.0
limited_casco_premium_100 -> 798.0
limited_casco_premium_200 -> 718.2
limited_casco_premium_500 -> 638.4
mtpl -> 400.0
DETECTED INCONSISTENCIES
-  limited_casco_compact_100: 820 -> 700.0
-  limited_casco_compact_200: 760 -> 630.0
-  limited_casco_compact_500: 650 -> 560.0
-  limited_casco_basic_100: 900 -> 700.0
-  limited_casco_basic_200: 780 -> 630.0
-  limited_casco_basic_500: 600 -> 560.0
-  limited_casco_comfort_100: 950 -> 749.0
-  limited_casco_comfort_200: 870 -> 674.1
-  limited_casco_comfort_500: 720 -> 599.2
-  limited_casco_premium_100: 1100 -> 798.0
-  limited_casco_premium_200: 980 -> 718.2
-  limited_casco_premium_500: 800 -> 638.4
-  casco_compact_100: 750 -> 900.0
-  casco_compact_200: 700 -> 810.0
-  casco_compact_500: 620 -> 720.0
-  casco_basic_100: 830 -> 900.0
-  casco_basic_200: 760 -> 810.0
-  casco_basic_500: 650 -> 720.0
-  casco_comfort_100: 900 -> 963.0
-  casco_comfort_200: 820 -> 866.7
-  casco_comfort_500: 720 -> 770.4
-  casco_premium_100: 1050 -> 1026.0
-  casco_premium_200: 950 -> 923.4
-  casco_premium_500: 780 -> 820.8
```

Basic unit tests

In [16]:

```
def test_rules():
        c = corrected_prices

        # Here, i will try to check if the products, in general, follow the rules t
        # I am trying to keep from the the PDF, i.e, if the minimun limited_casco i
        # than the mtpl_price and the minimun price for casco_ is bigger than the m
        # for limited_casco.

        mtpl_price = c["mtpl"]
        min_limited = min(j for i, j in c.items() if i.startswith("limited_casco_")
        min_casco = min(j for i, j in c.items() if i.startswith("casco_"))
        assert mtpl_price < min_limited < min_casco

        # I am going to test if the variant rule is alright for the product limited
        # for example, i.e, basic/compact < comfort < premium.

        base_100 = min(
            c["limited_casco_compact_100"],
            c["limited_casco_basic_100"],
        )
        comfort_100 = c["limited_casco_comfort_100"]
        premium_100 = c["limited_casco_premium_100"]
        assert base_100 < comfort_100 < premium_100

        # I am going to check if the deductible rule is alright for the product lim
        # for example, i.e, 100<200<500

        b100 = c["limited_casco_basic_100"]
        b200 = c["limited_casco_basic_200"]
        b500 = c["limited_casco_basic_500"]
        assert b100 > b200 > b500

test_rules()
print("Everything is correct")
```

```
Everything is correct
```