Flourishing Report

Martino Fusai

October 2023

Index

List of Tables

List of Figures

Introduction

My report focuses on the exploration and analysis of a dataset named `Flourishing_248_HD.xlsx`, a dataset that collects the data from 248 employees of a company that wishes to study their work-life balance. To do so, after an in-depth analysis of the data, I will subsequently implement different machine learning models.

0 Importing the Dataset & Preprocessing

First of all, I imported all the most important packages and the dataset itself, transforming it in a DataFrame (table 1).

	ID	age	education	sex	famstatus	pro_cat	pro_quant	priv_cat	priv_quant	positivity	flow
0	344	72	6	2	2	3	82	3	83	4.142857	44
1	317	45	6	2	2	3	81	3	83	4.000000	44
2	444	43	5	1	1	3	79	3	78	5.000000	47
3	270	60	2	2	3	3	79	3	82	4.666667	42
4	52	49	2	2	2	3	77	3	73	2.285714	33
5	...	...	...	...	...	...	...	...	...	...	...
243	242	37	6	1	3	1	22	2	43	1.650000	21
244	318	31	6	2	3	1	21	2	35	0.384615	21
245	502	29	6	2	1	1	39	1	25	0.527778	33
246	98	59	5	1	2	1	31	1	28	0.585366	26
247	26	39	3	2	3	1	43	3	63	1.125000	34

Table 1: `df` output [1]

This dataset encompasses a set of variables that provide a comprehensive understanding of the participants and their life contexts. The variable `ID` represents a unique identification number assigned to each participant, while the `sex` variable denotes the gender of the participants, with a value of 1 for males and a value of 2 for females. `famstatus` gives insights about the participants' family status, allowing for distinctions between those who are single, separated or divorced, and those in a couple. Then, articipants' educational background is captured by the `education` variable, reflecting their level of education, ranging from not completing a high school degree to completing at least 5 years of education beyond high school. The participants' age is represented by the `age` variable. The variables `prof_cat` and `priv_cat` reflect the mental well-being of the participants in professional and private contexts, with categories indicating whether they are languishing (if 1), moderately mentally healthy (if 2), or flourishing (if 3). `positivity` measures the ratio of positive to negative emotionsi. If the ratio is lesser than 1, the individual is diagnosed as depressed ii; between 1 and 2, the individual is languishing; then, between 2 and 3, the individual is moderately emotionally healthy iv; and finally, if the ratio is greater than 3, the individual is emotionally flourishing Furthermore, the `prof_quant` and `priv_quant` variables quantify mental well-being in these specific contexts. Lastly, the variable `flow` assesses the participants' flow level, encompassing aspects such as performance and concentration.

After setting the column `ID` as index of the DataFrame, I proceeded to replace the numerical values in the categorical variables with the corresponding status, and reordered to have a cleaner view of the situation.

Finally, I transformed all the categorical variables in a `category` datatype, while leaveing all the quantitative variables as `int64` or `float64`.

```
object_columns = df.select_dtypes(include=["object"]).columns                [8]

pd.options.mode.chained_assignment = None

for column in object_columns:
    df[column] = df[column].astype("category")
print(df.dtypes)

pd.options.mode.chained_assignment = "warn"
```

1 Univariate Data Analysis

Once finished the data preprocessing, I began analyzing the variables and the data.

1.1 Categorical Variables

I used the `.select_dtypes()` to select all the categorical variables, and by using `.describe()` I obtained a summary table of these categorical variables (table 2).

```
categorical_variables = df.select_dtypes(include=["category"])               [10]
summary_table = categorical_variables.describe()
```

	sex	education	famstatus	pro_cat	pos_cat
count	248	248	248	248	248
unique	2	6	3	3	4
top	female	5y or more after HS	couple	moderately mentally healthy	languishing
freq	151	110	173	159	143

Table 2: `print(summary_table)` output [10]

Then, I proceeded to plot their pie and bar charts (fig 1).

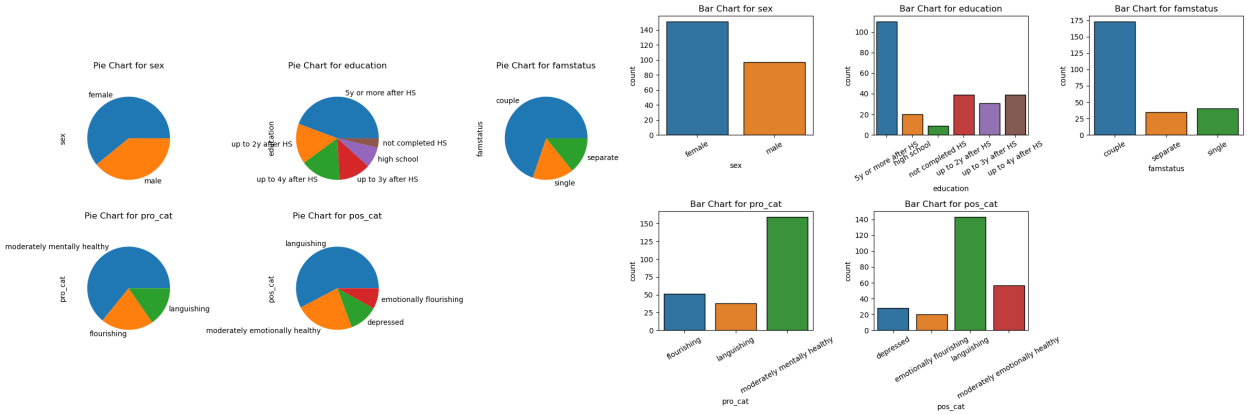

Figure 1: `categorical_variables` pie [11] and bar [12] charts

The `sex` variable displays a balanced distribution, with 151 instances of females and 97 instances of males, suggesting a relatively even gender representation. Educational backgrounds vary across six categories, whereby `5 years or more after high school` emerges as the predominant modality, featuring in 110 instances. Regarding family status, there is a prominent presence of `couple` (173 occurrences), while `pro_cat` is primarily dominated by `moderately mentally healthy`, which is observed 159 times. Finally, `languishing` is the most frequent category (143 instances) in the `positivity` variable.

1.2 Quantitative Variables

Same as for categorical variables, I created a new DataFrame containing all the quantitative variables of the original dataset.

Then, I created a grid of 2 rows and 3 columns of histogram plots for those variables. It loops through `quantitative_variables`, creating a histogram for each variable with 20 bins (2). The x-axis is labeled with the variable name, while the y-axis represents the frequency of occurrences. The code also removes the last subplot (`axes[5]`) from the grid using `fig.delaxes(axes[5])`, as it would have been empty. I then repeated the same process for the box plots as well (2).

```
fig, axes = plt.subplots(2, 3, figsize=(12, 7))                    [14]
axes = axes.flatten()

for i, column in enumerate(quantitative_variables):
    ax = axes[i]
    df[column].plot(kind="hist", ax=ax, bins=20, edgecolor="k")
    ax.set_title(f"Histogram for {column}")
    ax.set_xlabel(column)
    ax.set_ylabel("Frequency")

fig.delaxes(axes[5])

plt.tight_layout()
plt.show()
```

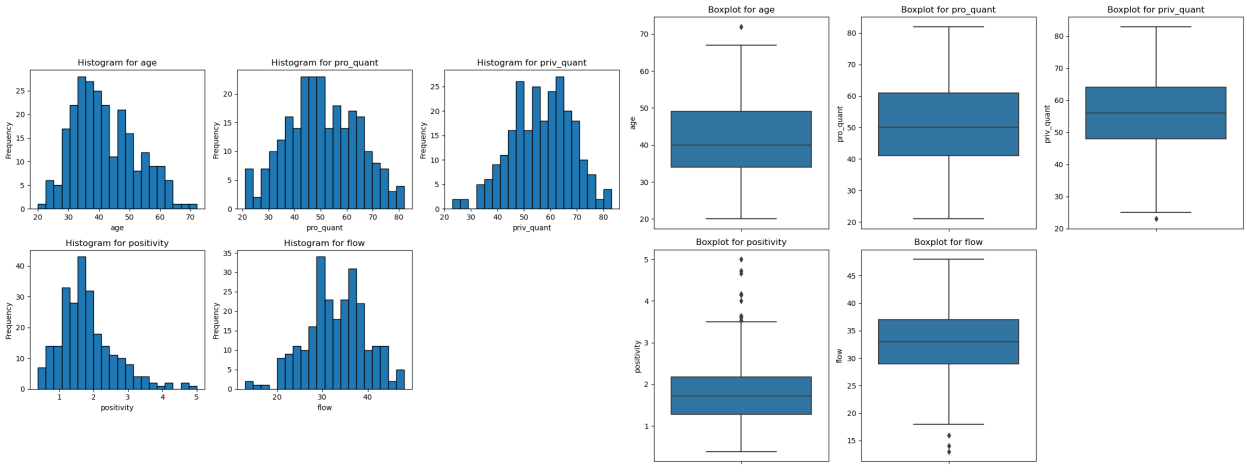

Figure 2: `quantitative_variables` histograms [14] and box plots [15]

2 Bivariate Data Analysis

I then moved on to the bivariate analysis, hoping to find important relations between some variables.

2.1 Both Categorical

If the variables are both categorical, I firstly calculated the contingency table.

```
for i, var1 in enumerate(categorical_variables):                    [16]
    for j, var2 in enumerate(categorical_variables):
        if i < j:
            contingency_table = pd.crosstab(df[var1], df[var2])
            print(f"Contingency Table for {var1} vs {var2}:
------------------\n{contingency_table}")
```

This code is using two nested loops to iterate through pairs of categorical variables in the dataset. For each unique pair (`var1`, `var2`), it calculates a contingency table using the `pd.crosstab` function. This contingency table summarizes the joint distribution of the two categorical variables, showing how many observations fall into each combination of categories.

Subsequently, I checked the significance of their relationship with the chi-square test and Cramer's V. The Chi-square statistic measures the strength of association between two categorical variables, and the p-value indicates the statistical significance of that association.

- For `sex` vs `education`: the Chi-square statistic is 1.825, with a high p-value of 0.873, suggesting that there is no significant association between these two variables.
- For `sex` vs `famstatus`: the Chi-square statistic is 8.261, with a low p-value of 0.016, indicating a significant association between these variables.
- For `sex` vs `pro_cat`: the Chi-square statistic is 0.466, with a high p-value of 0.792, suggesting no significant association.
- For `sex` vs `pos_cat`: the Chi-square statistic is 4.903, with a moderately high p-value of 0.179, implying once again no strong association.
- For `education` vs `famstatus`: the Chi-square statistic is 10.217, with a moderate p-value of 0.422, indicating no strong association.
- For `education` vs `pro_cat`: the Chi-square statistic is 11.947, with a p-value of 0.289, suggesting no strong association.
- For `education` vs `pos_cat`: the Chi-square statistic is 26.870, with a p-value of 0.030, indicating a significant association.
- For `famstatus` vs `pro_cat`: the Chi-square statistic is 2.992, with a high p-value of 0.559, suggesting no significant association.
- For `famstatus` vs `pos_cat`: the Chi-square statistic is 4.976, with a high p-value of 0.547, implying no significant association.
- For `pro_cat` vs `pos_cat`: the Chi-square statistic is 93.427, with an extremely low p-value (almost zero), indicating a highly significant association between these variables.

I then proceeded, with the backing of ChatGPT, to plot the side-by-side and stacked bar charts.

2.2 Both Quantitative

For the pairs of quantitative variables, I compute the correlation table (table 3) and plotted the heatmap to visualize the result (fig 3).

	age	pro_quant	priv_quant	positivity	flow
age	1.000000	0.229625	0.151607	0.257054	0.367994
pro_quant	0.229625	1.000000	0.635679	0.603502	0.536229
priv_quant	0.151607	0.635679	1.000000	0.636000	0.409439
positivity	0.257054	0.603502	0.636000	1.000000	0.439782
flow	0.367994	0.536229	0.409439	0.439782	1.000000

Table 3: `corr_table` output [20]

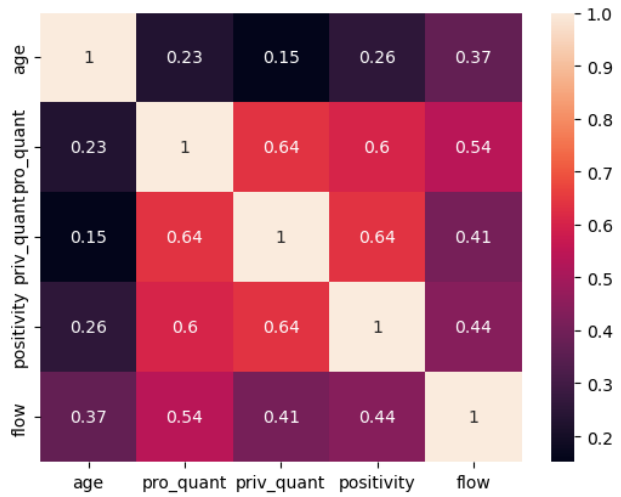

Figure 3: `sns.heatmap(corr_table, annot=True)` output [21]

The most significant correlations are between `priv_quant`, `positivity` and `pro_quant` (0.604), and between `age` and `flow` (0.368) (older individuals tend to experience a better work-life balance).

Subsequently, I computed again the correlation coefficient, and the p-value.

```
from scipy.stats import pearsonr                                    [22]

for i, var1 in enumerate(quantitative_variables):
    for j, var2 in enumerate(quantitative_variables):
        if i < j:
            variable1 = df[var1]
            variable2 = df[var2]
            r, p = pearsonr(variable1, variable2)
            print(f"Correlation coefficient (r) between {var1} and {var2}:
------------------{r}")
            print(f"p-value between {var1} and {var2}: {p}")
```

Finally, I plotted the scatterplots of the paired quantitative variables (fig 4). Specifically, I created a 5x5 grid of subplots using, and then looped through pairs of quantitative variables using two nested `for` loops, where `var1` and `var2` represent two different quantitative variables. The condition `if i < 5` and `j < 5` ensures that it only creates plots for the first 5 quantitative variables to avoid unnecessary duplication. Then, inside the loop, I used Seaborn's `scatterplot` function to create scatter plots for each pair of variables.

```
fig, axes = plt.subplots(5, 5, figsize=(12, 12))                    [23]

for i, var1 in enumerate(quantitative_variables):
    for j, var2 in enumerate(quantitative_variables):
        if i < 5 and j < 5:
            ax = axes[i, j]
            sns.scatterplot(data=df, x=var1,y=var2, ax=ax, color="lightblue")
            ax.set_title(f"{var1} vs {var2}")

plt.tight_layout()
plt.show()
```

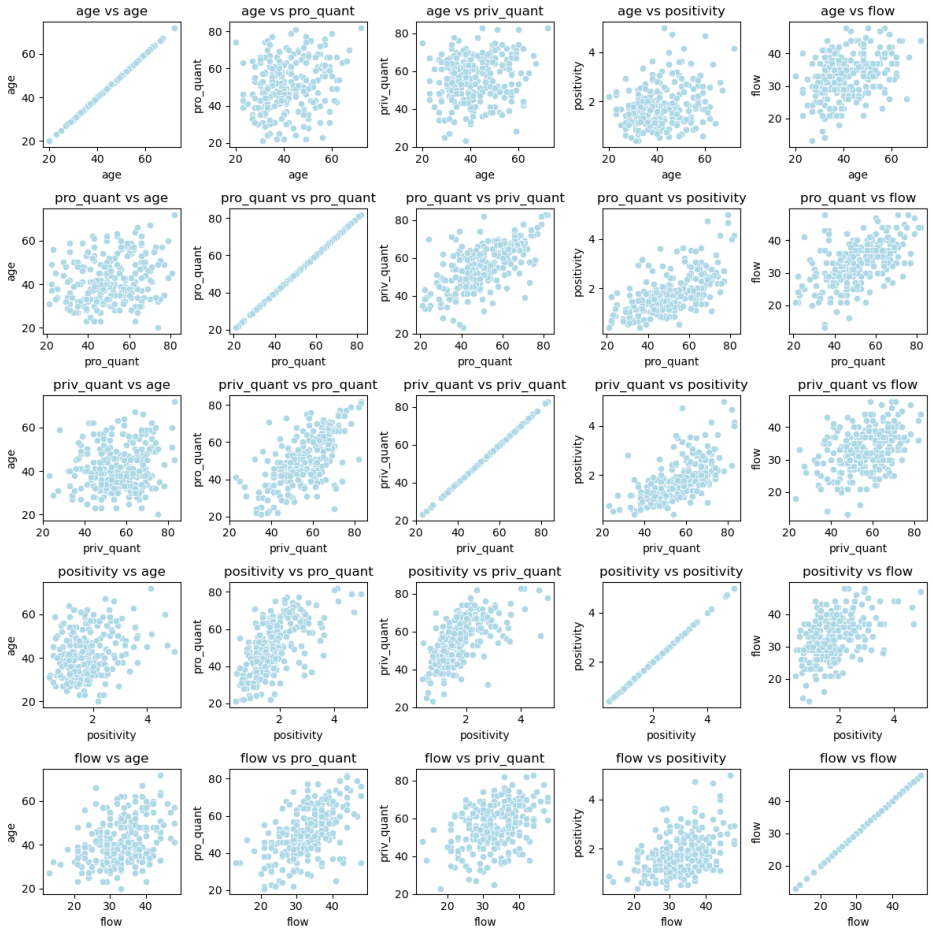

Figure 4: Scatter plots of paired quantitative variables [23]

2.3 Categorical & Quantitative

After, I performed an analysis of variance (ANOVA) to test for significant differences between categories of a categorical variable and a quantitative variable, followed by creating grouped box plots for visualization.

I used a nested loop structure to iterate through pairs of categorical and quantitative variables. Within the innermost loop, I initialized an empty list called `anova_results` to store the results of the ANOVA for

each category of the categorical variable; it iterates through the unique categories of the categorical variable (`df[cat_column].unique()`); then, for each category, it filters `df` to obtain the data for that specific category (`category_data`); finally, it appends a tuple containing the category name and the corresponding quantitative data to the `anova_results` list. After collecting data for each category, I calculated the ANOVA statistics, specifically the F-statistic and the p-value. I've done this using the `stats.f_oneway` function with the data from the `anova_results` list. The F-statistic assesses the variation between groups relative to the variation within groups. The code then prints out these results for the current pair of variables, including the F-statistic and the p-value. This gives us information about whether there are significant differences in the means of the quantitative variable among the different categories of the categorical variable. Next, I used Seaborn to create a grouped box plot (`sns.boxplot`) with the categorical variable on the x-axis and the quantitative variable on the y-axis. This plot visually represents the distribution of the quantitative variable for each category.

```
from scipy import stats                                                    [24]

for cat_column in categorical_variables:
    for quant_column in quantitative_variables:
        anova_results = []
        for category in df[cat_column].unique():
            category_data = df[df[cat_column] == category][quant_column]
            anova_results.append((category, category_data))

        f_statistic, p_value = stats.f_oneway
                                    (*[result[1] for result in anova_results])

        print(f"ANOVA for {cat_column} vs {quant_column}:")
        print(f"F-statistic: {f_statistic}")
        print(f"P-value: {p_value}")

        plt.figure(figsize=(5, 4))
        sns.boxplot(x=cat_column, y=quant_column, data=df)
        ax.set_title(f"Grouped Boxplot for {cat_column} vs {quant_column}")
        plt.title(f"Grouped Boxplot for {cat_column} vs {quant_column}")
        plt.xticks(rotation=30)

        plt.show()
```

3 Modeling

In this section, I will begin by implementing supervised learning models, followed by the application of unsupervised learning techniques.

I started from setting the predictors and the outcome variable, as follows.

```
X = quantitative_variables                                                 [25]
y = df["flow"]
```

I've then split the data into training and testing sets using the `train_test_split` function from scikit-learn. The `test_size` parameter is set to 0.2, which means 20% of the data will be used for testing, and the remaining 80% for training. The `random_state` parameter is set to 42 to ensure reproducibility.

```
from sklearn.model_selection import train_test_split                    [26]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=42)
```

In my analysis, I'll first train a machine learning model using the `X_train` and `y_train` datasets. Then, I'll make predictions on the testing data with this trained model using `X_test`. Finally, I'll evaluate the model's performance using appropriate metrics to determine its accuracy and effectiveness in making predictions.

3.1 Supervised Learning (Regression)

Regression focuses on predicting continuous numerical values as output. It leverages labeled data to learn patterns and relationships between input features and the target variable, allowing the model to make predictions about real-valued outcomes.

In order to predict the outcome variable based on the predictors, I used Linear Regression and K-Nearest Neighbors Regression.

3.1.1 Linear Regression

I've used the scikit-learn library to create a Linear Regression model, that finds a linear relationship between the input features and the target variable. This model minimizes the sum of squared differences between the predicted values and the actual target values.

Once the model is trained, I made predictions on the test set using the `predict` method. The predicted values are stored in the `y_pred` variable.

```
from sklearn.linear_model import LinearRegression                       [27]

lr = LinearRegression()
lr.fit(X_train, y_train)

y_pred = lr.predict(X_test)
```

To evaluate the model's performance, I'm using three common regression metrics (table 4). The MAE (*MeanAbsoluteError*) is a measure of the average absolute errors between the predicted and actual values. The MSE (*MeanSquaredError*) calculates the average of the squared differences between predicted and actual values. The R-squared (R^2) value measures the goodness of fit of the model.

```
from sklearn.metrics import mean_absolute_error, mean_squared_error,    [28]
                            r2_score

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Linear Regression Metrics:")
print(f"MAE: {mae}")
print(f"MSE: {mse}")
print(f"R-squared: {r2}")
```

Metric	Value
MAE	$7.673861546209083 \times 10^{-15}$
MSE	$1.040034217443382 \times 10^{-28}$
R-squared	1.0

Table 4: `LinearRegression` metrics [28]

Considering the dataset's small size with only 248 rows, the performance of the linear regression model is surprising. While the metrics appear impressive, we have to be cautious about such results. With a limited dataset, there's a risk of overfitting, where the model might have learned the training data almost by heart, resulting in an unrealistically high performance on this specific dataset. It's possible that the model won't generalize as well to unseen data or real-world scenarios. For a more robust evaluation, it would be necessary to test the model on a larger and more diverse dataset.

3.1.2 KNN Regression

I then moved on to using a K-Nearest Neighbors regression model. KNN Regression is a supervised machine learning technique used for predicting continuous numerical values. It does so by finding the `k` nearest data points in the training dataset to the input data point and calculating their average or weighted average to make predictions.

I created a `KNeighborsRegressor` object with a number of neighbors (`n_neighbors`) set to 5 and trained it on the training data using the fit method. Subsequently, I used the trained model to make predictions on a set of test data with the predict method.

```
from sklearn.neighbors import KNeighborsRegressor                    [29]

knn = KNeighborsRegressor(n_neighbors=5)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
```

Same as for Linear Regression, I then calculated the performance metrics for the KNN regression model (table 5).

```
mae = mean_absolute_error(y_test, y_pred)                            [30]
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"K-Nearest Neighbors Regressor Metrics:")
print(f"MAE: {mae}")
print(f"MSE: {mse}")
print(f"R-squared: {r2}")
```

Metric	Value
MAE	2.388
MSE	9.612
R-squared	0.7731

Table 5: `KNN Regressor` metrics [30]

With a MAE of 2.388, it means that, on average, the KNN regression model's predictions differ from the actual values by approximately 2.388 units. This indicates a reasonably good level of accuracy.
The MSE value of 9.612 indicates that the model's predictions have some variability from the true values. This result is moderately good.
The R-squared value of 0.7731 suggests that about 77.31% of the variance in the dependent variable (`y`) is explained by the independent variables used in the model. In other words, the KNN regression model does a decent job of capturing the underlying patterns in the data, although there is still some unexplained variance.

3.2 Supervised Learning (Classification)

The goal of classification is to predict a categorical outcome or class label based on a set of input features. It involves training a model on labeled data, where the correct outcomes are known, to learn the underlying patterns and relationships.

In particular, I used Logistic Regression and K-Nearest Neighbors Classification.

3.2.1 Logistic Regression

Logistic Regression models the probability of a data point belonging to a particular class using a logistic (S-shaped) function.

I created a logistic regression model by instantiating the `LogisticRegression` class with the parameter `max_iter` set to 2000, which increases the number of iterations for the solver to converge. Next, I trained the model using the training data by calling the fit method with `X_train` (the feature data) and `y_train` (the corresponding target labels). After training, I used the trained model to make predictions on the test dataset and stored the results in the `y_pred` variable. The `max_iter` parameter is adjusted to ensure the model converges during training.

```
from sklearn.linear_model import LogisticRegression                    [31]

logistic_model = LogisticRegression(max_iter=2000)
logistic_model.fit(X_train, y_train)

y_pred = logistic_model.predict(X_test)
```

First, I calculated the accuracy of the model's predictions using the `accuracy_score` function. Accuracy measures the proportion of correct predictions made by the model on the test dataset. Next, I computed the confusion matrix using the `confusion_matrix` function. A confusion matrix is a table that describes the model's predictions in terms of true positives, true negatives, false positives, and false negatives. The `classification_report` function generates a detailed report, including precision, recall, F1-score, and support for each class (table 6). The `zero_division` parameter is set to 1 to control the behavior of the report in cases where there are no true positives or false positives. This helps to prevent undefined metric warnings.

```
from sklearn.metrics import accuracy_score, classification_report,          [32]
                            confusion_matrix

accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(classification_report(y_test, y_pred, zero_division=1))
print('Confusion Matrix:\n', confusion)
```

	Precision	Recall	F1-Score	Support
Accuracy			0.20	50
Macro Avg	0.46	0.39	0.38	50
Weighted Avg	0.51	0.20	0.49	50

Table 6: `LogisticRegression` classification report [32]

The model's accuracy is 20%, indicating that it correctly predicts the class labels for one-fifth of the instances in the test dataset. To gain a deeper understanding, it's important to look at precision, recall, and the F1-score. Precision measures the model's ability to make accurate positive predictions. Recall, on the other hand, measures the model's ability to identify true positive instances. The F1-score is the harmonic mean of precision and recall. It provides a balanced measure that combines both precision and recall into a single value. Support indicates the number of instances for each class in the test dataset. It's important to keep in mind that the dataset is a bit imbalanced, meaning that some classes have significantly fewer instances than others, and this might have influenced the performance metrics. The `macro avg` and `weighted avg` values offer summaries of the model's overall performance. The macro average treats all classes equally, while the weighted average considers the impact of each class based on its size. The macro average F1-score is 0.38, and the weighted average F1-score is 0.49, indicating the overall model performance.

In conclusion, the logistic regression model's performance varies widely across different classes, and there's room for improvement, especially for classes with low precision and recall.

3.2.2 KNN Classification

KNN is another supervised learning method used for classification tasks. In particular, it classifies data points based on the class of their nearest neighbors in the feature space.

I created an instance of the `KNeighborsClassifier` class and set the number of neighbors to 5. This means that when making a prediction, the algorithm will consider the class labels of the five nearest data points in the training set. As usual, I trained the KNN classifier using the training data with the `.fit()` method. This step involves finding the nearest neighbors in the training data and learning how to make predictions based on their labels. Once the model is trained, I used it to make predictions on the test data. The predicted labels are stored in the `y_pred` variable.

```
from sklearn.neighbors import KNeighborsClassifier                          [33]

knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)

y_pred = knn_model.predict(X_test)
```

Just as before for Logistic Regression, I computed the performance metrics for KNN (table 7).

```
accuracy = accuracy_score(y_test, y_pred)                                    [34]
confusion = confusion_matrix(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(classification_report(y_test, y_pred, zero_division=1))
print('Confusion Matrix:\n', confusion)
```

	Precision	Recall	F1-Score	Support
Accuracy			0.14	50
Macro Avg	0.37	0.27	0.47	50
Weighted Avg	0.36	0.14	0.67	50

Table 7: `KNN Classifier` classification report [34]

The KNN classifier's performance appears to be limited, achieving low accuracy and less than optimal precision and recall, even worse than the Logistic Regression model.

3.3 Unsupervised Learning

In this section, I will implement unsupervised learning models. Unsupervised learning is a machine learning paradigm where the algorithm is provided with data that lacks explicit labels or target outcomes. Unlike supervised learning, where the model learns to predict an output variable based on input data, unsupervised learning focuses on discovering patterns, structures, or relationships within the data itself. In particular, I will utilize Principal Component Analysis (PCA). PCA is a dimensionality reduction technique that can help us reduce the number of features in the dataset while retaining as much of the variance as possible. PCA is primarily designed for continuous numerical variables, therefore I applied it on the `categorical_variables` DataFrame.

3.3.1 Principal Component Analysis (Rows)

I applied PCA to the rows of the dataset to reduce the dimensionality and capture the most important patterns. First, I created a `StandardScaler` instance called `scaler` to standardize the data. I did this to ensure that variables with different scales don't dominate the analysis. Then, I applied the `fit_transform` method to standardize the data. The `quantitative_variables.T` represents the transpose of the dataset, which means I performed PCA on the rows instead of columns.

After, I specified the number of principal components I wanted to retain, in this case, `n_components` set to 2, for visualization purposes. In other words, I aimed to reduce the data to just two principal components. Subsequently, I created a PCA instance `pca` with the specified number of components and I used the `fit_transform` method on the standardized data to calculate the principal components. The result was stored in `principal_components`. Finally, I created a new DataFrame `principal_df` to hold the principal components, naming the columns as `PC1` and `PC2` to represent the first and second principal components.

```
from sklearn.decomposition import PCA                                    [35]
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_data = scaler.fit_transform(quantitative_variables.T)

n_components = 2
pca = PCA(n_components=n_components)

principal_components = pca.fit_transform(scaled_data)

principal_df = pd.DataFrame(data=principal_components, columns=["PC1","PC2"])
```

After performing PCA as described above, I accessed the explained variance ratios of each principal component using the `explained_variance_ratio_` attribute of the PCA object. This provides information about how much of the original data's variance is explained by each principal component. Then, I stored the explained variance ratios in the `explained_variance` variable. This gives us a list of values, one for each principal component, indicating the proportion of the total variance explained by that component. To understand how much cumulative variance is explained as we include more principal components, I calculated the cumulative sum of the explained variances using the `cumsum()` method on the `explained_variance` variable (table 8).

```
explained_variance = pca.explained_variance_ratio_                       [36]
cumulative_variance = explained_variance.cumsum()
```

	Explained Variance Ratio	Cumulative Explained Variance
PC1	0.90245618	0.90245618
PC2	0.05286627	0.95532245

Table 8: `PC1` and `PC2` explained variances [36]

The explained variance ratio indicate the proportion of the total variance in the original data that each principal component explains. `PC1` explains approximately 90.25% of the total variance, while `PC2` explains about 5.29% of the total variance. The cumulative explained variance provides insights into how much of the total variance is explained when I include multiple principal components. The first value, 0.90245618, represents the cumulative explained variance when only the first principal component (`PC1`) is considered. The second value, 0.95532245, represents the cumulative explained variance when both `PC1` and `PC2` are considered. In this case, including both `PC1` and `PC2`, they collectively explain approximately 95.53% of the total variance. `PC1` is the dominant principal component and explains a significant portion of the variance in the data, suggesting that it captures the most critical patterns or features. Therefore, I could also decide to keep as principal component just the first one.

I then proceeded to create a scree plot, that is a graphical representation of the explained variance for each principal component in a Principal Component Analysis (fig 5).

```
plt.plot(range(1, len(explained_variance) + 1), cumulative_variance,
         marker="o", linestyle="--")
plt.title("Scree Plot")
plt.xlabel("Number of Principal Components")
plt.ylabel("Cumulative Explained Variance")
plt.show()
```
[37]

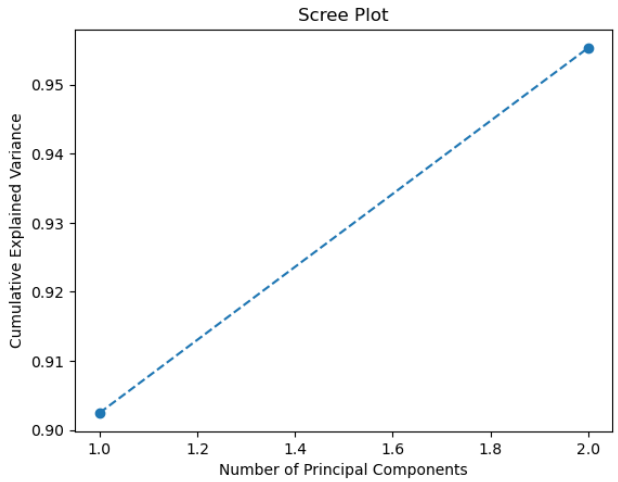

Figure 5: PCA scree plot [37]

To conclude, I'm specifically interested in understanding the loadings of the two principal components. First, I accessed the `pca.components_` attribute, which contains the principal components. Each row in this array represents a principal component, and each column corresponds to the original features in my dataset. Next, I used array indexing `[:2, :]` to select the first two rows (`PC1` and `PC2`) and all columns (all the original features). This allowed me to focus on the loadings of `PC1` and `PC2`. I stored these loadings in a variable called `loadings`, which essentially contains the information about how each original feature contributes to the variance captured by `PC1` and `PC2`, and then printed them out. The loadings are like weights or coefficients that indicate the strength and direction of the relationship between each original feature and the principal components. Positive loadings mean a positive correlation, while negative loadings indicate a negative correlation. `PC1` and `PC2` are linear combinations of the original features, and the loadings tell us how each feature contributes to these principal components.

```
loadings = pca.components_[:2, :]
print("Loadings of PC1 and PC2:")
print(loadings)
```
[38]

`PC1` is primarily influenced by features with negative loadings, while `PC2` has a mix of both positive and negative influences.

3.3.2 Principal Component Analysis (Variables)

I then repeated the same process for the columns of the dataset. The only difference in the code is that I applied the `.fit_transform` method to standardize the data on the original dataset and not the transposed one.

I proceeded to calculate the explained variance ratio and cumulative explained variance for `PC1` and `PC2` on the columns (table 9).

	Explained Variance Ratio	Cumulative Explained Variance
PC1	0.55569293	0.19125766
PC2	0.55569293	0.74695059

Table 9: `PC1` and `PC2` explained variances [40]

When combined, `PC1` and `PC2`, explain a total of approximately 74.70% of the variance in the data. This means that `PC1` captures the most significant variation, while `PC2` captures a smaller but still substantial portion of the remaining variation.

Finally, I generated the scree plot (fig 6) and the loadings (table 10).

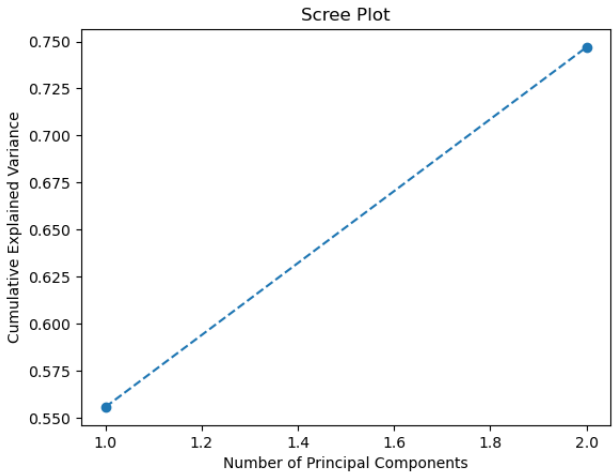

Figure 6: PCA scree plot [41]

	age	pro_quant 2	priv_quant 3	positivity	flow
PC1	0.2692	0.5074	0.4821	0.4927	0.4415
PC2	-0.8485	0.1775	0.3614	0.1859	-0.2887

Table 10: `PC1` and `PC2` loadings [42]

Overall, the positive loadings suggest that the variables move in a similar direction within PC1, which is a linear combination of the original variables. However, PC2 captures a mix of variables where age has the most significant influence but in the opposite (negative) direction.

Conclusion

In summary, in this report I explained the coding and the analysis process that I've conducted on the `Flourishing_248_HD.xlsx` dataset, and the machine learning models I implemented, in particular Linear Regression and KNN Regressor for the prediction, Logistic Regression and KNN Classifier for the classification, and Principal Component Analysis for unsupervised learning. In my opinion, it would be necessary a larger dataset to draw some reliable and clear conclusions.