An End-to-End Machine Learning Workflow for Salary Prediction

Can salaries really be predicted from data, or are they influenced by many other factors?

---

by Carlos Torres Gonzalez

October 2025

1. Introduction

The tech sector in Norway is growing fast, pushed forward by digitalization, innovation, and the need for skilled developers. With this growth, salaries have become an important measure of value in the market. Understanding what drives salaries, can help employers who want to attract talent, policymakers who aim for fairness, and employees who want to understand their own career paths.

1.1 Background and Motivation

When we talk about salaries in Norway, pay can differ depending on region, industry, type of work, and gender. By using machine learning on survey data, this project aims to explore these differences and also build models that can predict annual salaries. The motivation is both, practical and meaningful: applying data science methods to real data and contributing to the conversation on pay and transparency.

Predicting salaries is an interesting topic for me because it connects directly with my own future. I am going to work in IT (if everything goes as planned), a sector that is constantly growing and where salaries might vary a lot depending on experience, specialization, or even the region where you work. What motivates me here is the opportunity to apply what I am learning in data science and machine learning to a subject that feels real and close to my own career path.

I also find it curious to test whether machine learning can actually predict something that is often seen as a **taboo** topic: the salaries. On one hand, salary can be viewed as a **tangible**, numerical value that depends on measurable factors such as education, years of experience, or region. On the other hand, it is also influenced by aspects that are a lot **harder to capture**, like your personality, negotiation skills, or even how pretty you are (there are many studies about it, for example: IZA World of labor, 2017).

With this project, I want to explore to what extent salaries are something "tangible" that can be modeled with data, or whether they go beyond numbers and reflect personal factors that are impossible to measure.

This project is also a way to demonstrate the skills I have gained throughout the Noroff course: cleaning and preparing data, visualizations, building models, and comparing their performance. It is a technical challenge, but at the same time, an opportunity to see for myself the possibilities and limits of machine learning when applied to real-world problems.

This dual motivation, both professional and personal, frames the project as an opportunity not only to apply machine learning methods, but also, to reflect on their value and limitations when addressing real-world problems.

1.2 Problem Statement

The problem I focus on here is to find out whether predicting developer salaries in Norway can really be as objective as it seems, using demographic and professional factors. The task is not simple because survey data can be messy, categories may have many different values, and there is always variation in human responses. The challenge is: to prepare the data properly, visualize and explore it, clean it, create models, and see how well they perform at capturing salary patterns.

I also want to find some hidden patterns on data by using **Matplotlib** (library from Python) for: visualization and comparing models.

1.3 Objectives

- Main Objective: Evaluate to what extent machine learning models can predict developer salaries in Norway, and whether salaries are influenced by additional factors beyond the dataset.
- Secondary Objectives:
    - Explore the dataset to find trends in salary by gender, region, experience, and other factors.

- Preprocess the data with encoding and scaling to prepare it for modeling.
- Train and test at least three machine learning models, comparing them with metrics such as MAE, RMSE, and R^2.
- Reflect on the dataset limits and what the results mean.

1.4 Scope and Limitations

This project uses a dataset of Norwegian developers collected through surveys and shared on Kaggle. The scope includes data cleaning, preprocessing, some feature engineering (not applied but explained), model training, and evaluation.
There are clear limits: the dataset is based on self-reported survey responses, which may include bias since it covers only one year (2024), so it cannot capture long-term salary dynamics or labor market changes; and it does not include all possible salary factors, such as company size or negotiation skills. These points need to be kept in mind when interpreting the results.

2. Background Literature / Domain Description

Salary prediction has long been explored in economics, human resources, and more recently, data science. Traditional research consistently shows that, salaries, are influenced by measurable factors such as education, years of experience, gender, geographic region, and type of employment contract. However, studies also highlight intangible factors—such as negotiation skills, company size, and even personal characteristics like appearance or networking ability, that make the prediction of salaries a complex problem. This combination of quantifiable and no-quantifiable drivers explains why salary prediction has been a subject of debate across both: social sciences and technical disciplines (Hamermesh & Biddle, 1994; Glassdoor, 2021).

In Norway, the technology sector has been expanding rapidly, creating strong demand for skilled developers. According to Statistics Norway (SSB, 2023), IT professionals remain among the top quartile of earners, with salaries significantly higher in urban regions such as Oslo and Viken compared to more peripheral areas. Reports from Tekna (2022) show similar

trends, noting that senior engineers, and developers too, earn well above the national average. Despite Norway's reputation for equality, studies continue to report a persistent gender pay gap, of around 12–15%, in technology-related jobs (OECD, 2022). Compairing with other Nordic countries, similar dynamics can be seen: Sweden and Denmark also show strong regional salary differences and evidence of gender disparities, even within highly skilled sectors (World Economic Forum, 2023). These regional, and demographic imbalances make Norway a particularly relevant case study for testing predictive models.

On the international level, surveys such as the annual **Stack Overflow Developer Survey** provide insight into how global developer salaries vary by education, experience, job type, and geographic region. For example, in 2023 edition, developers specializing in machine learning and data science reported higher-than-average compensation compared to web development or system administration (Stack Overflow, 2023). Such findings reinforce the importance of including professional field and specialization as predictive features.

Machine Learning has increasingly been applied to the problem of wage prediction. Regression-based methods have historically been the most common approach, starting from simple linear regression models that test the impact of education and experience on income (Mincer, 1974). More recent studies have expanded this to no-linear models: Random Forests and Gradient Boosting have been successfully used to capture complex interactions between demographic and also professional variables (Sharma & Panwar, 2020). Support Vector Regression and Neural Networks have also been tested, although results often show that tree-based ensembles tend to outperform them on structured tabular data (Mousa et al., 2022). Across these studies, a recurring conclusion is: that machine learning models can capture patterns and improve predictive accuracy, but their performance is always limited by the quality, representativeness, and completeness of the available data.

One important limitation reported in the literature is the reliance on **self-reported surveys**, which are subject to both recall bias and deliberate misreporting (Groves et al., 2011). In addition, categorical features with many unique values (such as geographic region or job field) create challenges for encoding, which can affect model performance if not handled carefully. Another limitation relates to generalization: while models may perform well within a given country or dataset, their predictive power drops when applied across different regions, industries, or even years. This shows that salary prediction is not only a technical task but also one deeply tied to the context of the data being used.

As a summary, prior research indicates that salary prediction is feasible to an extent, but inherently imperfect. Models can explain part of the variance (often around 40–60% depending on the dataset and method) but cannot capture the full complexity of human labor markets. For this project, the aim is to build upon this body of literature by testing whether survey data from Norwegian developers can be modeled effectively using different algorithms, while reflecting critically on the limitations identified in previous studies.

3. Dataset Description

3.1 Data Source

The dataset used in this project, as we mentioned before, comes from Kaggle and is based on survey responses from developers in Norway. The survey collected information about demographics, education, work situation, and salaries. The file was provided in CSV format and imported into Python using Pandas. Since the original column names were in Norwegian, I translated them into English only to make the code clearer and more familiar, not because it was strictly necessary.

3.2 Data Structure

I import the dataset into a pandas DataFrame using **pd.read_csv()** function (see ***Snippet 1***).

I check the shape of the dataset to see how many rows and columns it contains: (2682 rows and 8 columns). Finally, I display the first 10 rows with **.head(10)** function to quickly inspect the structure and content of the data.

This confirms the dataset includes demographic and work-related information such as *gender, education, experience, region, employment type, field, salary, and bonus.*

```
[1]: import pandas as pd
[2]: df = pd.read_csv('/Users/carlostorres/Desktop/salaries.csv')
[3]: df.shape
[3]: (2682, 8)
[4]: df.head(10)
```

[4]:

	kjønn	utdanning	erfaring	arbeidssted	arbeidssituasjon	fag	lønn	bonus?
0	mann	4	7	Agder	in-house, privat sektor	AI / maskinlæring	865000	Nei
1	mann	5	2	Nordland	in-house, offentlig/kommunal sektor	AI / maskinlæring	756000	Nei
2	mann	4	30	Oslo	konsulent	AI / maskinlæring	1500000	Nei
3	mann	5	25	Oslo	konsulent	AI / maskinlæring	1200000	Nei
4	mann	9	18	Oslo	in-house, privat sektor	AI / maskinlæring	1200000	Nei
5	mann	2	14	Oslo	in-house, privat sektor	AI / maskinlæring	839000	Nei
6	kvinne	5	9	Oslo	konsulent	AI / maskinlæring	1200000	Ja
7	mann	6	9	Oslo	in-house, privat sektor	AI / maskinlæring	980000	Ja
8	annet / ønsker ikke oppgi	5	5	Oslo	in-house, privat sektor	AI / maskinlæring	750000	Nei
9	mann	5	5	Oslo	konsulent	AI / maskinlæring	910000	Nei

Snippet 1. Using shape() and head() function to see data structure.

The dataset contains 2,682 rows and 8 columns where each row represents one survey response. The key variables include:

- **gender**: respondent's gender
- **education_level**: level of education completed (numeric scale)
- **experience_years**: number of years of work experience
- **region**: Norwegian region where the respondent works
- **employment_type**: type of job (consultant, in-house private, in-house public, etc.)
- **field**: main professional field (e.g., AI/machine learning, web development)
- **salary**: annual salary in NOK
- **bonus**: whether the respondent receives a bonus

These variables include both numerical and categorical data, which makes preprocessing a necessary step before applying machine learning models.

3.3 Data Quality

Initial checks (we'll go though the code later) showed that the dataset contained no missing values in the key columns and no obvious duplicates after cleaning. However, some potential challenges were identified:

- **Survey bias**: data is self-reported, which may introduce errors or exaggerations.
- **Category imbalance**: the number of men on the dataset is much higher than women, which may affect results.
- **High cardinality on some variables**: variables such as region and field have many unique values, making encoding proccess more complex.

Overall, the dataset is relatively clean and ready for analysis, but these quality issues need to be kept in mind when interpreting the results.

4. Exploratory Data Analysis (EDA)

The exploratory data analysis (EDA) stage is an essential step in any data science project because it allows us to move beyond raw numbers and start uncovering patterns, trends, and potential issues in the dataset.

At this stage, I relied on the Matplotlib library to produce visualizations that make the information easier to interpret. By plotting distributions, comparisons, and summaries, it becomes possible to see how salaries vary across different groups and how they relate to important factors such as gender, education, years of experience, region, and employment type. These visual insights provide the foundation for making informed decisions about data preprocessing and model selection later in the project.

- **Distribution of salaries**: Figure 1 shows that most salaries fall within a middle range, with fewer very low or very high values.

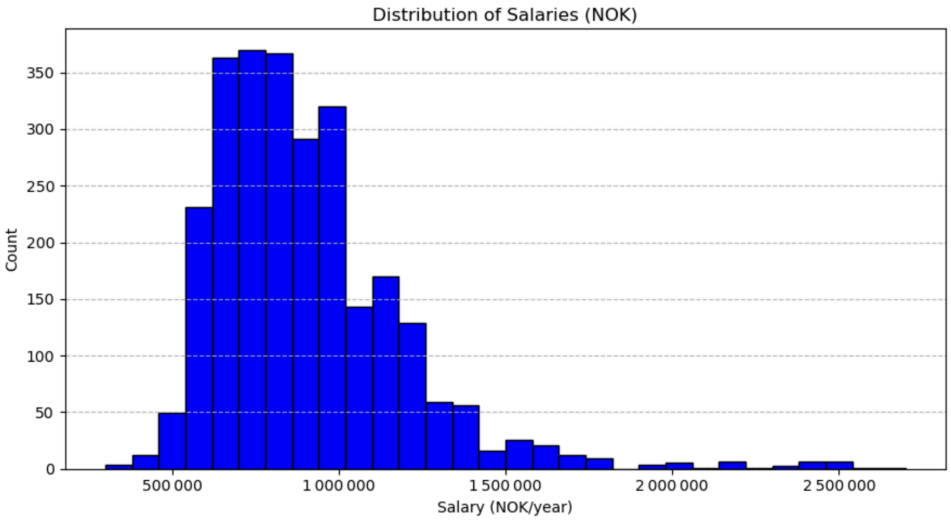

Figure 1. Salaries distribution by NOK/year.

- **Gender differences**: Average salary is higher for men than for women, and the dataset contains many more men than women (as shown in Figure 2), which reflects an imbalance in the tech workforce, something we could not see without visualization.

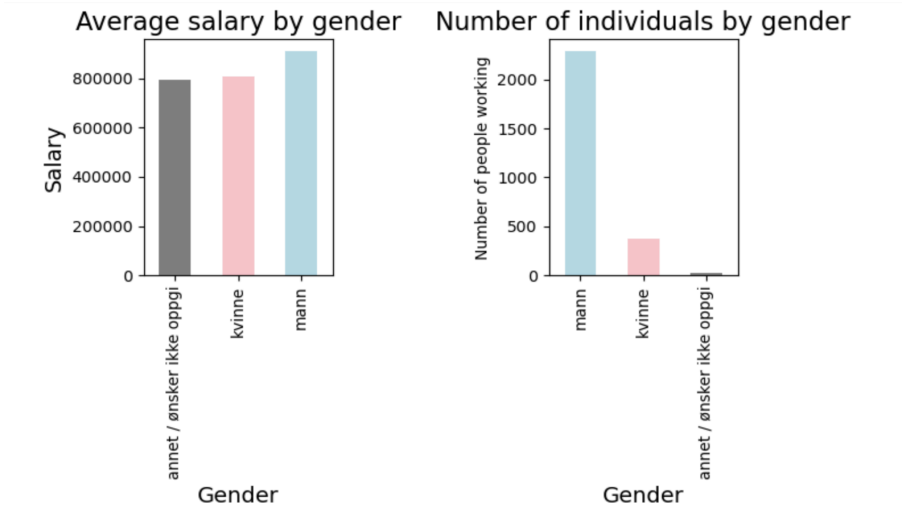

Figure 2. Comparing average salary by gender. Also the number of men/women/others working in the industry.

- **Regional variation**: Figure 3 shows that the salaries are highest in Oslo and Viken, while other regions show lower averages, reflecting the concentration of tech jobs and companies.

Figure 3. Salaries in Norway based on Region.

- **Experience**: Salaries generally increase with years of experience (see Figure 4), although the relationship is not perfectly linear due to outliers and some noisy data.

Figure 4. How salary increases based on years of experience.

- **Bonus**: Respondents who receive a bonus also show higher average salaries (as shown in Figure 5). Something that we could expect:

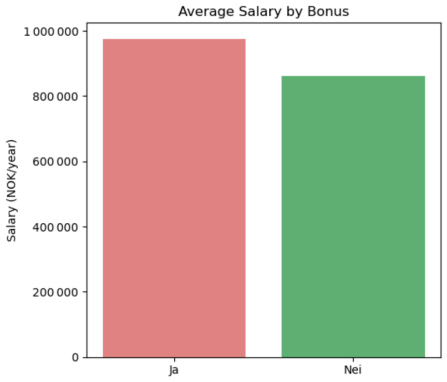

Figure 5. Salary based on bonuses.

- **Field of work**: Average salaries vary depending on the professional field (see Figure 6), with some areas such as AI and machine learning showing higher values compared to others.

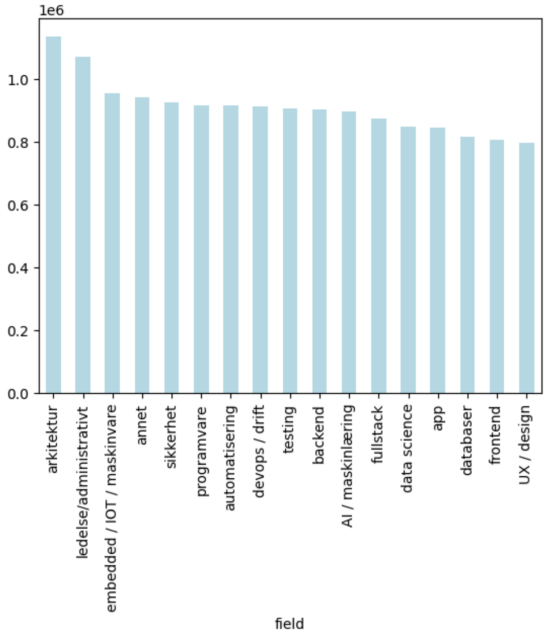

Figure 6. Fields in the IT world and respective salaries.

These first insights highlight important trends and also suggest potential biases, such as gender imbalance and regional concentration. They also show that no single factor fully explains salary differences, which underlines the value of building models that consider multiple features together.

5. Data Preprocessing

Data preprocessing is a crucial step in any machine learning workflow, as the quality of the input data directly impacts the performance of the models. Raw data often contains issues such as missing values, duplicates, inconsistencies, or irrelevant information that can lead to biased or inaccurate results if not handled properly. By applying preprocessing techniques, we ensure that the dataset is clean, consistent, and ready for analysis. This step also includes preparing variables in a way that makes them suitable for the algorithms used later in the project, improving both efficiency and reliability of the predictions.

5.1 Data Cleaning

The first step is shown in Snippet 2, the idea was to create a copy of the dataframe and then to check for missing values and duplicates. So we use the .copy() function and we save the copy in a variable. We use isnull().sum() function to get the total number of null values, which is zero in this case:

```
[19]: df2 = df.copy()

[20]: df2.isnull().sum()

[20]: gender             0
      education_level    0
      experience_years   0
      region             0
      employment_type    0
      field              0
      salary             0
      bonus              0
      dtype: int64

[21]: df2.drop_duplicates(inplace=True)
```

Snippet 2. Using copy() function to create a copy of the dataframe. isnull().sum() to check null values and drop_duplicates() to eliminate all duplicate data.

We apply **drop_duplicates().** function to delete all possible duplicates. Inplace = **True** so the changes are made on the same variable.

The dataset showed no missing values in the main variables and duplicates were removed. This confirmed that the data was mostly clean and ready for further processing.

5.2 Categorical Encoding

Since the dataset contained several categorical variables, encoding was necessary.

I apply one-hot encoding (check Snippet 3) to categorical variables with a small number of categories (*gender*, *employment_type*, and *bonus*) because I do not want to get many columns on the output.

Using a **for loop**, each variable is transformed into new binary columns (0/1) using **OneHotEncoder()** (Called OHE to make it simple):

```
[23]: # Automatize the encoding by using a For loop.
variables_ohe = ['gender', 'employment_type', 'bonus']

for column in variables_ohe:
    onehot = OneHotEncoder(dtype=int,
                           sparse_output=False)
    encoded= onehot.fit_transform(df2[[column]])
    encoded_df = pd.DataFrame(encoded, columns=onehot.get_feature_names_out([column]), index=df2.index)
    df2= pd.concat([df2, encoded_df], axis=1)
    df2.drop(columns=column, inplace=True)
```

Snippet 3. One Hot Encoding by using a For Loop.

These new columns are added back to the dataset, while the original categorical column is removed. This step ensures that machine learning models can interpret these variables numerically, without imposing an artificial order.

Variables with many unique values, like *region* and *field*, were encoded with ordinal encoding to keep the dataset manageable. Snippet 4 shows how I used the same **for loop** but using an **OrdinalEncoder()** to encode '*region'* and '*field*' variables:

```
[24]: # Automatize the encoding by using again a For loop.
      variables_oe = ['region', 'field']

      for column in variables_oe:
          encoder = OrdinalEncoder(dtype=int)
          encoded = encoder.fit_transform(df2[[column]])
          df2[column] = encoded
```

Snippet 4. Ordinal Encoding by using a For Loop.

5.3 Feature Scaling

Snippet 5 shows how education level and years of experience variables were put on the same scale using **StandardScaler**():

```
[25]: from sklearn.preprocessing import StandardScaler
      scaler = StandardScaler()
      num_cols = ["education_level", "experience_years"]
      df2[num_cols] = scaler.fit_transform(df2[num_cols])
```

Snippet 5. Applying Standard Scaler to 'education_level' and 'experience_years' columns.

This ensured that differences in scale would not affect the models, especially those sensitive to the magnitude of values.

5.4 Final Dataset Shape

After cleaning, encoding, and scaling, the dataset ended up with **14 features** ready for modeling. With the transformation done, the dataset is ready for the models.

Snippet 6 demonstrates how I checked the shape of the Dataset already pre-processed one more time using **.shape** method:

```
[86]: df2.shape
[86]: (2645, 14)
```

Snippet 6. Using the .shape method to double check and finish pre-processing block.

The new dataset went from 2682 rows to 2645 and from 8 columns to 14.

Pre-processing is done, we can jump ahead to the Feature Engineering step.

6. Feature Engineering

Some feature engineering ideas were considered here (check Snippet 7), but I decided not to apply them in the final models because they did not affect the model performance.

First idea was grouping years of experience into categories such as Entry, Junior, Mid, Senior, and Expert, second idea was applying a log transformation to the salary variable to reduce skewness.

[21]:
```
# Feature engineering ideas ( not used to train the models):

# 1. Group 'experience_years' into categories

# df2["experience_level"] = pd.cut(
#     df2["experience_years"],
#     bins=[0, 2, 5, 10, 20, 40],
#     labels=["Entry", "Junior", "Mid", "Senior", "Expert"]
# )


# 2. Log-transform the target variable 'salary'

# df2["salary_log"] = np.log1p(df2["salary"])
```

Snippet 7. Some Feature engineering ideas not used but tested.

These ideas were noted but the dataset was kept in its original form for simplicity and clarity.

7. Model Development

Model development is the stage where the prepared dataset is used to train and evaluate different machine learning algorithms. The goal is to build predictive models that can generalize well to unseen data, rather than just memorizing the training examples. This process involves splitting the data into training and testing sets, selecting appropriate algorithms, and evaluating their performance using suitable metrics. By following a structured approach, we can compare different models fairly and identify which one provides the most reliable predictions for the salary dataset.

7.1. Train/Test Split

The dataset was split into 80% for training and 20% for testing. A fixed random state was used to make the results reproducible (see Snippet 8).

Here, the function **train_test_split** was imported, X and y were designed and then I printed the shape of both Train and Test groups, after splitting, to check that everything was correct.

```
[89]: from sklearn.model_selection import train_test_split

[90]: X = df2.drop(columns=['salary'])
      y = df2['salary']

[91]: X_train, X_test, y_train, y_test  = train_test_split(X, y, test_size=0.2, random_state=42)

[92]: print(f'Train shape: {X_train.shape}')
      print(f'Test shape: {X_test.shape}')

      Train shape: (2116, 13)
      Test shape: (529, 13)
```

Snippet 8. Defining X and y, using train_test_split() function to split the data into train/test.

7.2. Model Selection

Model selection is a key step in the machine learning workflow, as different algorithms capture different types of relationships within the data. Choosing a diverse set of models allows us to compare their strengths and weaknesses in terms of predictive accuracy, interpretability, and ability to handle non-linear patterns. For this project, **three models** were selected: **Linear Regression, Random Forest Regressor,** and **Support Vector Regressor (SVR)**. These were chosen to represent different approaches: linear, ensemble-based, and kernel-based, ensuring a fair comparison between simple, flexible, and more advanced methods.

7.2.1. Linear Regression

General description: Linear Regression is one of the most fundamental supervised learning algorithms. It assumes a linear relationship between the input features and the target variable, estimating coefficients that minimize the Residual Sum of Squares (RSS). Its simplicity

makes it easy to interpret and widely used as a baseline model in predictive tasks. The principle of Linear Regression is illustrated in Figure 7.

Reason for selection: It was chosen as a starting point because it serves as a clear baseline: fast to train, highly interpretable, and useful for identifying linear patterns in the salary data. Even though it may not capture complex relationships, it provides a benchmark against which more advanced models can be compared.

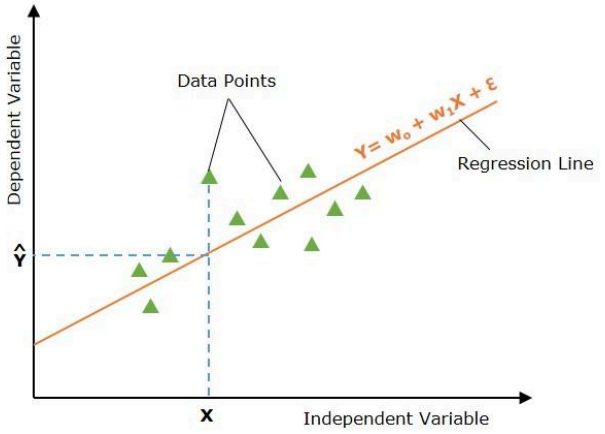

Figure 7. Illustration of how Linear Regression fits a line to minimize the distance between predicted and actual values.

7.2.2. Random Forest Regressor

General description: Random Forest is an ensemble method based on decision trees. It constructs multiple trees using bootstrapped samples of the data and random subsets of features, then averages their predictions (the principle of Random Forest Regression is illustrated in Figure 8). This process reduces variance and improves stability compared to a single tree. Random Forest handles both categorical and numerical variables and can capture non-linear relationships effectively.

Reason for selection: It was included because it is a robust model that performs well on **heterogeneous tabular data** such as the salary dataset, which contains both numerical and categorical features. Additionally, it provides feature importance scores, which can be used to identify which variables most strongly influence salaries.

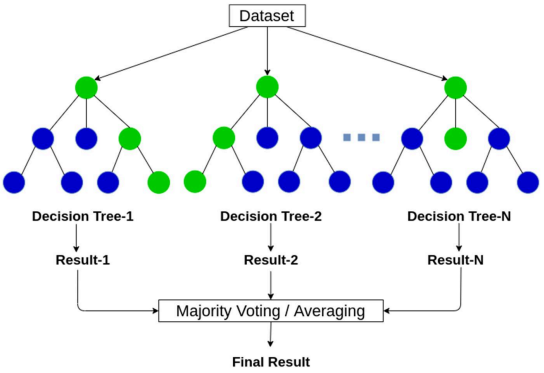

Figure 8. Illustration of how Random Forest Regression combines predictions from multiple decision trees to reduce variance and improve accuracy.

7.2.3. Support Vector Regressor (SVR)

General description: Support Vector Regression is based on the principles of Support Vector Machines (SVM). Instead of trying to minimize the residuals directly, SVR attempts to fit the best line (or hyperplane) within a margin of tolerance, known as the epsilon margin (see visual example on Figure 9). Predictions within this margin are considered acceptable, while errors outside of it are penalized. SVR can use kernel functions (such as linear, polynomial, or radial basis functions) to capture non-linear relationships between features and the target variable.

Reason for selection: It was chosen because it provides a different approach compared to tree-based and linear models. By using kernels, SVR can model non-linear relationships in the salary dataset and is particularly useful when the data is not linearly separable. It adds diversity to the set of models evaluated, ensuring that different types of patterns in the data are considered.

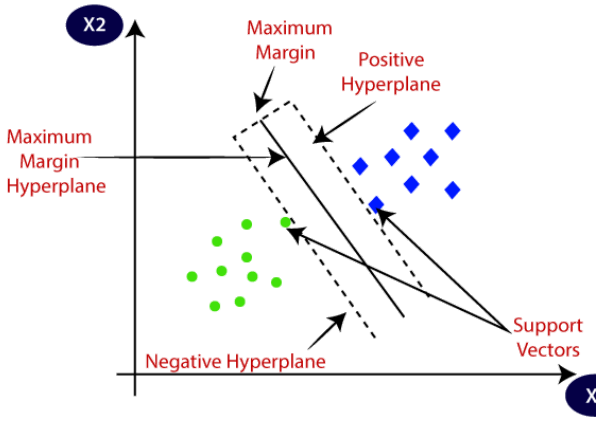

Figure 9. Illustration of Support Vector Regression, where a tolerance margin (maximum margin) is defined around the regression line and only errors outside this margin influence the model.

7.3. Model Training

Each model was trained on the training set and then evaluated on the test set. To make this process clear and consistent, I created **two** helper functions.

The first function (see Snippet 9) handled model training by fitting the model to the training data and returning it:

```
[94]: def model_training(model, X_train, y_train):
          model.fit(X_train, y_train)
          return model
```

Snippet 9. First function that returns the model already trained.

The second function (see Snippet 10) took a trained model, made predictions on the test set, and then calculated evaluation metrics such as MAE, RMSE, and R^2 and print them on the output:

```python
def predict_evaluate(model, X_test, y_test, model_name="name of the model"):

    y_pred = model.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)

    print(f"\n*** {model_name} model evaluation ***\n")
    print(f" \t- MAE : {mae:.0f}")
    print(f" \t- RMSE: {rmse:.0f}")
    print(f" \t- R²  : {r2:.2f}")
```

Snippet 10. Second function that prints evaluation metrics from each model.

This approach avoided repeating code three times and provided a structured way to compare model performance.

8. Model Performance Evaluation

8.1. Evaluation Method Selection

Evaluating model performance is a fundamental step in machine learning, as it allows us to measure how well the models generalize to unseen data. Using appropriate evaluation metrics ensures that the results are not only accurate but also meaningful and interpretable in the context of the problem. In this project, I selected a set of commonly used regression metrics—Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and the Coefficient of Determination (R^2)—to provide a balanced view of model accuracy, error magnitude, and explanatory power.

8.1.1. Mean Absolute Error (MAE)

General description: Mean Absolute Error measures the average magnitude of prediction errors, without considering whether they are positive or negative. It is calculated as the mean of the absolute differences between predicted and actual values, expressed in the same units as the target variable.

Reason for selection: For salary prediction, MAE clearly shows the **typical prediction error** in Norwegian Kroner (NOK), making it intuitive for stakeholders to interpret how far predictions deviate from real salaries on average.

8.1.2. Root Mean Squared Error (RMSE)

General description: Root Mean Squared Error is the square root of the average of squared differences between predictions and actual values. By squaring errors before averaging, RMSE penalizes large deviations more strongly than MAE.

Reason for selection: Because salary data often contains extreme values, RMSE is important for capturing how well the model performs not only on typical cases but also on high-end outliers. It emphasizes the cost of large mistakes in salary prediction.

8.1.3. Coefficient of Determination (R^2)

General description: R^2, or the coefficient of determination, measures how much of the variance in the target variable can be explained by the model. It ranges from negative values (worse than predicting the mean) to 1 (perfect prediction).

Reason for selection: R^2 was included because it provides a **normalized measure of explanatory power**, independent of the salary scale. It allows comparing models based on how much structure in the salary data they capture, rather than on raw error magnitudes.

8.2. Model Evaluation

As mentioned before, the performance of each model was evaluated using Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R^2. The results show clear differences between the models:

- #### 8.2.1. Linear Regression

Snippet 11 presents the code used for training and evaluating the Linear Regression model:

Linear Regression Model

```
[125]: linear_trained = model_training(LinearRegression(), X_train, y_train)
       predict_evaluate(linear_trained, X_test, y_test, model_name='Linear Regression')

       *** Linear Regression model evaluation ***

              - MAE : 157744
              - RMSE: 237759
              - R²  : 0.40
```

Snippet 11. Code for training and evaluating the Linear Regression model.

- MAE: 157,744 NOK
- RMSE: 237,759 NOK
- R^2: 0.40

Interpretation:

This baseline model explains only 40% of the variance in salaries, indicating that linear assumptions are not sufficient to capture the complexity of the data. The relatively high error values also suggest that more flexible models are needed to better represent non-linear relationships in the dataset.

- **8.2.2. Random Forest Regressor**

Snippet 12 presents the code used for training and evaluating the Random Forest Regressor model.

Random Forest Regressor model

```
[126]: rforest_trained = model_training(RandomForestRegressor(), X_train, y_train)
       predict_evaluate(rforest_trained, X_test, y_test, model_name='Random Forest Regressor')

       *** Random Forest Regressor model evaluation ***

              - MAE : 136691
              - RMSE: 214245
              - R²  : 0.51
```

Snippet 12. Code for training and evaluating the Random Forest Regressor model.

- MAE: 136,691 NOK
- RMSE: 214,245 NOK
- R^2: 0.51

Interpretation:

Compared to Linear Regression, Random Forest performs better by capturing more complex patterns and reducing prediction errors. With an R^2 of 0.51, it explains just over half of the variance in salaries, indicating improved performance but also showing that significant unexplained variability remains.

- **8.2.3. Support Vector Regressor (SVR)**

Snippet 13 presents the code used for training and evaluating the Support Vector Regressor (SVR) model:

SVM model

```
[127]: svm_trained = model_training(SVR(), X_train, y_train)
       predict_evaluate(svm_trained, X_test, y_test, model_name='SVM')

       *** SVM model evaluation ***

              - MAE : 215581
              - RMSE: 312950
              - R²  : -0.05
```

Snippet 13. Code for training and evaluating the Support Vector Regressor (SVR) model.

- MAE: 215,581 NOK
- RMSE: 312,950 NOK
- R^2: -0.05

Interpretation:

The SVR model performed poorly in this case, failing to generalize and even producing a negative R^2 score. This indicates that the model performs worse than simply predicting the mean salary. While SVR can be effective in some regression tasks, in this dataset it is not well suited due to the complexity and scale of the features.

These results suggest that while **Random Forest** provides the most reliable predictions, even the best model leaves a **large margin of error**. This underlines the complexity of salary prediction and the influence of many factors not included in the dataset.

9. Model Optimization

Model optimization is the step where algorithms are fine-tuned to achieve better performance. While the initial models provide a baseline, they often rely on default parameters that may not be optimal for the dataset. By systematically adjusting hyperparameters, we can reduce errors, improve accuracy, and ensure that the models generalize better to unseen data. This stage is therefore essential for moving from a basic implementation to a more robust and reliable predictive solution.

9.1. Model Tuning

In this stage, I applied hyperparameter tuning to improve model performance using **RandomizedSearchCV.** The process involved testing multiple parameter combinations with cross-validation, and then selecting the best configuration based on validation results. This optimization was carried out for both the **Random Forest** and the **Support Vector Regressor (SVR).**

I created a helper function (eval_model, refer to Snippet 14) to evaluate all models in a consistent way. The function uses the model's predictions on the test set to calculate three key metrics: MAE, RMSE, and R^2.

It prints the results in a clear format and also returns the values, which helps avoid repeating code and makes it easier to compare different models fairly:

```python
def eval_model(name, model, X_te, y_te):
    y_pred = model.predict(X_te)
    mae  = mean_absolute_error(y_te, y_pred)
    rmse = mean_squared_error(y_te, y_pred, squared=False)
    r2   = r2_score(y_te, y_pred)
    print(f"\n{name}")
    print(f" MAE : {mae:,.0f}")
    print(f" RMSE: {rmse:,.0f}")
    print(f" R²  : {r2:.3f}")
    return mae, rmse, r2
```

Snippet 14. Helper function to evaluate all models.

- **Random Forest tuning**

In the first part I apply hyperparameter tuning to the Random Forest model using **RandomizedSearchCV**. Different values are tested for parameters such as the number of trees (n_estimators), tree depth (max_depth), minimum samples required for a split (min_samples_split), and other settings (as shown in Snippet 15). The search is done by using a cross-validation method, and the best-performing combination is selected. Finally, the tuned model is evaluated on the test set to compare its performance against the baseline Random Forest:

```
# Random Forest TUNING
rf = RandomForestRegressor(random_state=42, n_jobs=-1)
param_rf = {
    "n_estimators": [200, 400, 600, 800, 1000],
    "max_depth": [5, 10, 20, None],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
    "max_features": ["sqrt", "log2", None]
}
rf_search = RandomizedSearchCV(
    rf, param_rf, n_iter=30, cv=cv, scoring=scoring,
    n_jobs=-1, random_state=42, verbose=0
)
rf_search.fit(X_train, y_train)
rf_best = rf_search.best_estimator_
print("Best RF params:", rf_search.best_params_)
eval_model("Random Forest (tuned)", rf_best, X_test, y_test)
```

Snippet 15. Random Forest Tuning with RandomizedSearchCV().

Best parameters: {n_estimators=200, min_samples_split=5, min_samples_leaf=1, max_features='log2', max_depth=10}

- MAE: ~134,745 NOK
- RMSE: ~212,175 NOK
- R^2: 0.52

This tuning **improved** the baseline Random Forest (MAE ~138,404, R^2 0.49) by reducing error and slightly increasing the explained variance.

- **SVR tuning**

In this second part, I optimize the Support Vector Regressor (SVR) using **RandomizedSearchCV** too (see Snippet 16)**.** The search explores different parameter combinations, including the penalty parameter C, the margin of tolerance epsilon, kernel

type, and gamma values. By testing multiple settings with cross-validation, the algorithm identifies the best-performing configuration. The tuned SVR model is then evaluated on the test set and compared with the baseline version to assess improvements:

```
# SVR TUNING
svr = SVR()
param_svr = {
    "C": [0.1, 1, 10, 100],
    "epsilon": [0.01, 0.1, 0.5, 1],
    "kernel": ["linear", "rbf"],
    "gamma": ["scale", "auto"]
}
svr_search = RandomizedSearchCV(
    svr, param_svr, n_iter=20, cv=cv, scoring=scoring,
    n_jobs=-1, random_state=42, verbose=0
)
svr_search.fit(X_train, y_train)
svr_best = svr_search.best_estimator_
print("Best SVR params:", svr_search.best_params_)
eval_model("SVR (tuned)", svr_best, X_test, y_test)
```

Snippet 16. SVR Tuning with RandomizedSearchCV().

Best parameters: {kernel='linear', gamma='auto', epsilon=0.5, C=100}

- MAE: 178,245 NOK
- RMSE: 278,522 NOK
- R^2: 0.17

This represents an **improvement over the baseline** SVR (MAE ~215,581, R^2 -0.05), but **performance is still lower** than both Linear Regression and Random Forest.

Overall, hyperparameter optimization shows that **Random Forest** benefits the most from tuning, **reaching the best performance** among the tested models. SVR improves compared to its baseline but the model is still weak.

These results confirm that salaries are influenced by many complex factors, and even the best-tuned models still leave a large prediction error.

9.2. Model Comparison: Baseline vs Tuned

In Table 1, we can observe that the comparison of models highlights both the potential and the limitations of salary prediction using machine learning. **Linear Regression** provided a simple baseline, explaining 40% of the variance with relatively high errors (MAE: 157,744 NOK). **Random Forest** clearly outperformed the baseline, reducing both MAE and RMSE

while increasing R^2 to 0.49. After hyperparameter tuning, the **Random Forest** model achieved the best overall performance, with MAE reduced to 134,745 NOK and R^2 increased to 0.52, showing that tree-based methods are well suited for this type of structured data. In contrast, the **Support Vector Regressor** struggled: the baseline SVR failed to generalize (R^2 = -0.05), and even after tuning, performance improved only modestly (R^2 = 0.17). This outcome was expected, since SVR is not an ideal algorithm for datasets of this size and structure, but it was included deliberately to demonstrate and confirm its limitations in this context.

Table 1. Comparison of model metrics (baseline vs tuned).

Model	MAE (NOK)	RMSE (NOK)	R^2
Linear Regression	157,744	237,759	0.40
Random Forest (baseline)	139,24	218,958	0.49
Random Forest (tuned)	134,745	212,175	0.52
SVR (baseline)	215,581	312,95	-0.05
SVR (tuned)	178,245	278,522	0.17

Figure 10 provides a comparison between the baseline and tuned versions of the Random Forest and Support Vector Regressor models:

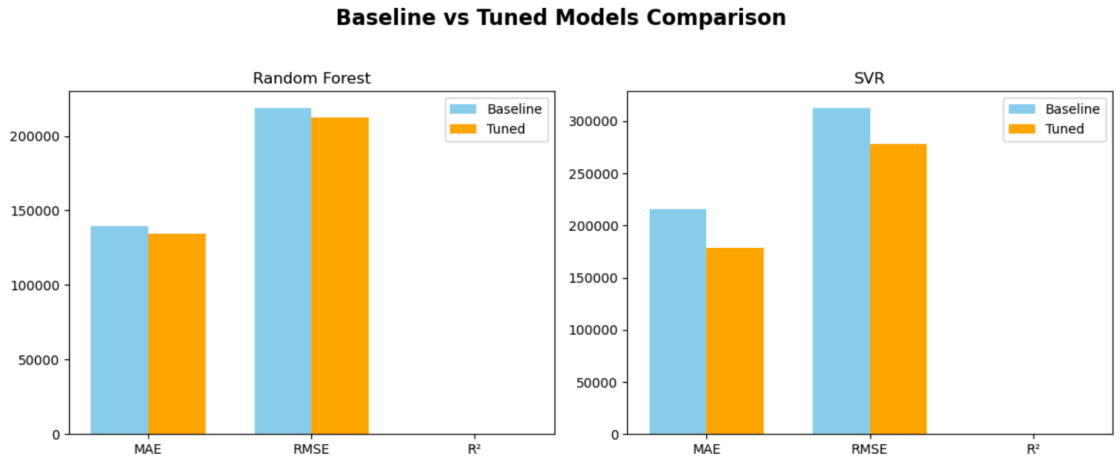

Figure 10. Comparison of baseline and tuned models (Random Forest and SVR) using MAE, RMSE and R^2.

Figure 10 shows the comparison between baseline and tuned models for Random Forest and SVR, using MAE, RMSE, and R^2 as evaluation metrics. The blue bars are the baseline results and the yellow bars are the tuned versions after hyperparameter optimization.
We can see that **tuning helps both models**: Random Forest reduces errors, and SVR also improves, especially in MAE and RMSE. Still, Random Forest is clearly the stronger model, giving better accuracy and stability overall. It is important to note that **R^2 does not appear** in this chart, because its **values are very small** compared to MAE and RMSE and are hidden on this scale. This plot shows both the value of tuning and the fact that some algorithms, like SVR, are not as well suited to this dataset as Random Forest.

While the previous figure focused mainly on MAE and RMSE, Figure 11 gives a more detailed view by including R^2 alongside the error metrics. Here we can clearly see how Random Forest tuning slightly increases R^2, while SVR tuning improves R^2 from a negative baseline but still remains weaker overall. This chart presents the three metrics side by side makes the comparison more complete and easier to interpret:

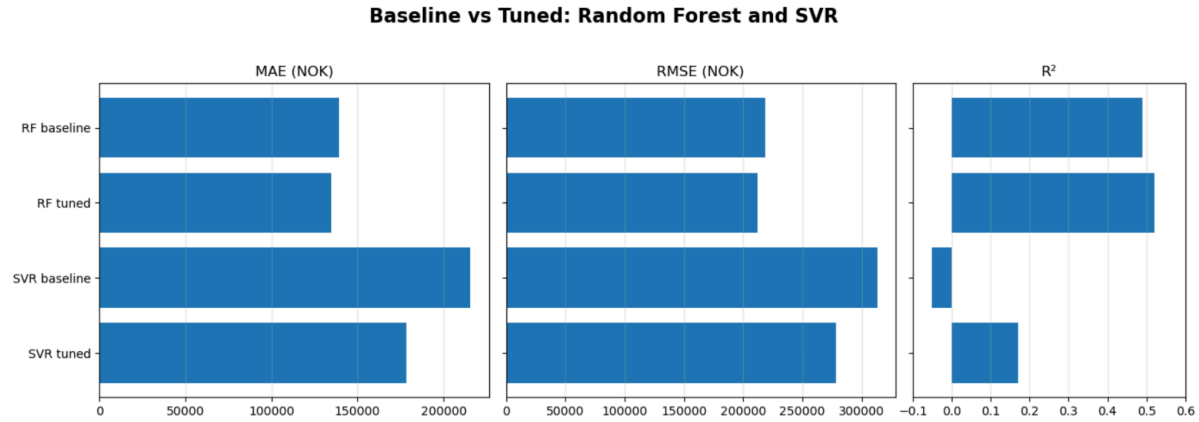

Figure 11. Comparing MAE, RMSE and R2 metrics.

These results suggest that while ensemble methods like Random Forest can capture more complex salary patterns, the dataset still leaves a large margin of error, reflecting the fact that salaries depend on many external and personal factors not captured by the survey.

10. Results and Discussion

The results highlight both the potential and the limits of using machine learning for salary prediction. The exploratory analysis showed clear patterns:

- Salaries are higher in Oslo and Viken (see Figure 3).
- Men earn more on average than women (see Figure 2).
- Bonuses are linked to higher average salaries (see Figure 5).
- Salaries increase with years of experience, although not perfectly linearly (see Figure 4).
- Some fields, such as AI and machine learning, are better paid than others (see Figure 6).

However, these patterns alone do not fully explain salary differences.

From the models, Random Forest proved to be the strongest, both in the baseline and after optimization. Its tuned version reached an R^2 of around 0.52, showing that it can explain about half of the variance in salaries. Linear Regression was weaker but still useful as a baseline, while SVR struggled even after tuning. This confirms that tree-based methods are more suited for this type of structured data.

Even with optimization, the errors remain large (MAE above 130,000 NOK). This shows that **salaries are influenced by many factors that cannot easily be captured in a dataset.** Elements such as company size, negotiation skills, or job role matter, but there are also more personal aspects like relationships at work, personality, or other human factors that go beyond what data can measure. This connects back to the main objective: testing whether salaries can be predicted in a straightforward way. The answer is partly yes, models can capture some structure, but the high error also shows that salary prediction is far from exact.

In short, these results show both the strengths and the limits of using machine learning in real salary problems. Models can find patterns, but they should not be read as exact predictions for each person. Research also shows that salaries can be shaped by many things we cannot measure in a dataset. Studies even talk about a "beauty premium," where people who are seen as more attractive, fit, or well-presented often earn more (Hamermesh & Biddle, 1994; Deryugina & Shurchkov, 2015; Wellesley College Study, 2021).

11. Conclusion and Reflection

This project has been a complete journey from raw survey data to machine learning models. The main lessons can be summarized as follows:

- **Objectives met**: The project successfully built a workflow to predict salaries and showed both the possibilities and the limits of machine learning in this context.
- **Insights from EDA**: Clear patterns appeared regarding gender, region, experience, and bonuses, but these factors alone cannot fully explain salary differences.
- **Model results**: Random Forest was the most effective model, improving after tuning, while Linear Regression gave a simple baseline and SVR struggled even after optimization.
- **Limits of prediction**: High errors remained, proving that salaries depend on many aspects that are not in the dataset and cannot easily be measured.
- **Personal reflection**: This project taught me how to handle data end-to-end, from cleaning and encoding to model evaluation and optimization. It also showed me that machine learning is powerful but must be applied with care and humility.

Overall, I feel the objectives were reached. I learned both technical skills and broader lessons about the complexity of real-world problems, which I will take with me as I continue studying and working in the IT and machine learning fields.

12. References

Deryugina, T., & Shurchkov, O. (2015). Now you see it, now you don't: The vanishing beauty premium. Journal of Economic Behavior & Organization, 116, 331–345.

Glassdoor Economic Research. (2021). What's behind the gender pay gap? Retrieved from https://www.glassdoor.com/blog/category/research/

Groves, R. M., Fowler, F. J., Couper, M. P., Lepkowski, J. M., Singer, E., & Tourangeau, R. (2011). Survey methodology (2nd ed.). John Wiley & Sons.

Hamermesh, D. S., & Biddle, J. E. (1994). Beauty and the labor market. American Economic Review, 84(5), 1174–1194.

IZA World of Labor. (2017). Does it pay to be beautiful? Retrieved from https://wol.iza.org/press-releases/does-it-pay-to-be-beautiful

Mincer, J. (1974). Schooling, experience, and earnings. Columbia University Press.

Mousa, A., Alshamsi, A., & Alzaabi, A. (2022). Machine learning approaches for salary prediction: A comparative study. Journal of Applied Data Science, 3(2), 45–60.

OECD. (2022). OECD employment outlook 2022. OECD Publishing. https://doi.org/10.1787/19991266

Sharma, A., & Panwar, A. (2020). Predicting employee salaries using machine learning techniques. International Journal of Computer Applications, 176(30), 6–12.

Stack Overflow. (2023). Developer survey 2023. Retrieved from https://survey.stackoverflow.co/

Statistics Norway (SSB). (2023). Wage statistics. Retrieved from https://www.ssb.no

Tekna. (2022). Salary statistics for engineers and technologists. Retrieved from https://www.tekna.no

World Economic Forum. (2023). Future of jobs report 2023. Retrieved from https://www.weforum.org/reports/future-of-jobs-2023