04 - Salary Prediction

In this notebook, we will:

- Load the cleaned HR dataset.
- Prepare and encode features for regression.
- Split the data into training and testing sets.
- Train a simple `linear regression` model.
- We will then use `GridSearchCV` to test different tree based models, including `GradientBoosting`, `RandomForest` and `ElasticNet`.
- If more than 1 model shows promising results, we will also try to use a **stacking regressor** to combine models.
- We will try to find the best model to predict **MonthlyIncome**.
- Display sample predictions to illustrate how the model performs.

Since MonthlyIncome is a continuous variable, this task is approached as a regression problem.

Import Libraries & Load Data

We start by importing the necessary libraries and loading the cleaned dataset (saved as "hr_data_cleaned.csv").

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Regression libraries
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

# Set plotting style
sns.set(style="whitegrid")
plt.rcParams["figure.figsize"] = (12, 8)

# Load the cleaned dataset
df = pd.read_csv("hr_data_cleaned.csv")
print("DataFrame shape:", df.shape)
df.head()
```

```
DataFrame shape: (1470, 36)

   Age Attrition     BusinessTravel  DailyRate              Department  \
0   41       Yes      Travel_Rarely       1102                   Sales
```

```
1   49        No  Travel_Frequently       279  Research & Development

2   37       Yes      Travel_Rarely      1373  Research & Development

3   33        No  Travel_Frequently      1392  Research & Development

4   27        No      Travel_Rarely       591  Research & Development


   DistanceFromHome  Education EducationField  EmployeeCount
EmployeeNumber  \
0                 1          2  Life Sciences              1
1
1                 8          1  Life Sciences              1
2
2                 2          2          Other              1
4
3                 3          4  Life Sciences              1
5
4                 2          1        Medical              1
7

   ...  StandardHours StockOptionLevel  TotalWorkingYears  \
0  ...             80                0                  8
1  ...             80                1                 10
2  ...             80                0                  7
3  ...             80                0                  8
4  ...             80                1                  6

   TrainingTimesLastYear  WorkLifeBalance YearsAtCompany
YearsInCurrentRole  \
0                      0                1              6
4
1                      3                3             10
7
2                      3                3              0
0
3                      3                3              8
7
4                      3                3              2
2

  YearsSinceLastPromotion  YearsWithCurrManager  TenureBucket
0                       0                     5           3-6
1                       1                     7          6-10
2                       0                     0           NaN
3                       3                     0          6-10
4                       2                     2            <3
```

```
[5 rows x 36 columns]
```

Simple Linear Regression with Preprocessing

Overview

We train a **Linear Regression model** to predict **MonthlyIncome**.

We use a pipeline

A **Pipeline** ensures smooth data preprocessing before training the model:

- **SimpleImputer**: Handles missing values in numerical features by filling them with the median. We actually do not have any missing values, hence we wont be using this.

- **OneHotEncoder**: Converts categorical features (university, degree, field of study) into numeric format for the model.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

# Target variable
target = "MonthlyIncome"

# Features
categorical_features = ["BusinessTravel", "Department",
"EducationField", "Gender","OverTime", "JobRole", "MaritalStatus"]
numerical_features = ["JobLevel","Age", "DistanceFromHome",
"Education", "NumCompaniesWorked", "TotalWorkingYears",
"TrainingTimesLastYear",
                      "YearsAtCompany", "YearsInCurrentRole",
"YearsSinceLastPromotion", "YearsWithCurrManager"]

# Drop rows where target is missing
df = df.dropna(subset=[target])

# Define preprocessing (One-Hot Encoding)
categorical_transformer = OneHotEncoder(handle_unknown="ignore")

full_transformer = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numerical_features),  # Scale
numerical features
        ("cat", categorical_transformer, categorical_features),
```

```
    ]
)

# Define Linear Regression model pipeline
linear_pipeline = Pipeline([
    ("preprocessor", full_transformer),
    ("model", LinearRegression())
])

# Train-test split
X = df[numerical_features + categorical_features]
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train the model
linear_pipeline.fit(X_train, y_train)

# Predictions
y_pred = linear_pipeline.predict(X_test)

# Evaluate model and print metrics

print(f"Linear Regression - Mean Squared Error:
{mean_squared_error(y_test, y_pred):.2f}")
print(f"Linear Regression - Mean Absolute Error:
{mean_absolute_error(y_test, y_pred):.2f}")
print(f"Linear Regression - R² Score: {r2_score(y_test, y_pred):.2f}")
```

```
Linear Regression - Mean Squared Error: 1367382.92
Linear Regression - Mean Absolute Error: 886.77
Linear Regression - R² Score: 0.94
```

GridSearchCV

- Exploring other models that possibly could be better than simple Linear Regression.

- Different models or better hyperparameters could improve performance.

Process

- Use `GridSearchCV` to test the following tree based models: `DecisionTree`,`GradientBoosting`,`RandomForest`.

- Since we are only using tree based models, there is no need for a StandardScaler. Additionally, we have no missing values, eliminating the need for a SimpleImputer.

- Tune `max_depth`, `n_estimators`, and `learning_rate`.

- We find the best parameters for each model to get the most optimised result.
- Compare results and select the best-performing model.

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor

# Target variable
target = "MonthlyIncome"

# Features
categorical_features = ["BusinessTravel", "Department",
"EducationField", "Gender","OverTime", "JobRole", "MaritalStatus"]
numerical_features = ["JobLevel","Age", "DistanceFromHome",
"Education", "NumCompaniesWorked", "TotalWorkingYears",
"TrainingTimesLastYear", "YearsAtCompany", "YearsInCurrentRole",
"YearsSinceLastPromotion", "YearsWithCurrManager"]

# Drop rows where target is missing
df = df.dropna(subset=[target])

# Define preprocessing (Handle missing values + One-Hot Encoding)
categorical_transformer = OneHotEncoder(handle_unknown="ignore")

full_transformer = ColumnTransformer(
    transformers=[
        ("cat", categorical_transformer, categorical_features),
    ]
)

# Define pipelines for different models
pipelines = {
    "DecisionTree": Pipeline([("preprocessor", full_transformer),
("model", DecisionTreeRegressor())]),
    "RandomForest": Pipeline([("preprocessor", full_transformer),
("model", RandomForestRegressor())]),
    "GradientBoosting": Pipeline([("preprocessor", full_transformer),
("model", GradientBoostingRegressor())])
}

# Define hyperparameter grids and find the best parameters
param_grids = {
    "DecisionTree": {
        "model__max_depth": [3, 5, 10, None],  # Limits depth of tree
(None = unlimited depth)
        "model__min_samples_split": [2, 5, 10],  # Minimum samples
required to split a node
```

```
        "model__min_samples_leaf": [1, 2, 5],  # Minimum samples
required at a leaf node
        "model__max_features": [None, "sqrt", "log2"]  # Number of
features to consider when splitting
    },
    "RandomForest": {
        "model__n_estimators": [100, 200, 300],
        "model__max_depth": [None, 10, 20],
        "model__min_samples_split": [2, 5, 10]
    },
    "GradientBoosting": {
        "model__learning_rate": [0.01, 0.05, 0.1],  # Lower values
prevent overfitting
        "model__n_estimators": [100, 200, 300],  # More trees improve
performance
        "model__max_depth": [3, 5, 7],  # Controls tree complexity
        "model__subsample": [0.7, 0.85, 1.0]  # Subsample reduces
variance
    }
}

# Train-test split (Ensure all years are represented)
X = df[numerical_features + categorical_features]
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Fit and evaluate each model using GridSearchCV
results = {}
for name, pipeline in pipelines.items():
    print(f"Training {name}...")
    grid_search = GridSearchCV(pipeline, param_grids[name], cv=3,
scoring="neg_mean_squared_error", n_jobs=-1)
    grid_search.fit(X_train, y_train)

    # Get best model
    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test)

    # Evaluate
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    results[name] = {"MSE": mse, "R²": r2, "Best Params":
grid_search.best_params_}

# Print results
for name, result in results.items():
    print(f"{name} Regression - MSE: {result['MSE']:.2f}, R²:
```

```
{result['R²']:.2f}")
    print(f"Best Parameters: {result['Best Params']}\n")

Training DecisionTree...
Training RandomForest...
Training GradientBoosting...
DecisionTree Regression - MSE: 4754752.82, R²: 0.78
Best Parameters: {'model__max_depth': 5, 'model__max_features': None,
'model__min_samples_leaf': 5, 'model__min_samples_split': 2}

RandomForest Regression - MSE: 5146368.90, R²: 0.76
Best Parameters: {'model__max_depth': 10, 'model__min_samples_split':
10, 'model__n_estimators': 300}

GradientBoosting Regression - MSE: 4575549.94, R²: 0.79
Best Parameters: {'model__learning_rate': 0.05, 'model__max_depth': 3,
'model__n_estimators': 100, 'model__subsample': 0.7}
```

Analysis:

- All models seem to perform similarly well after mutiple attempts at trial and testing the pest possible parameters for each model.
- However simple linear regression seems to be a substantially better model.
- Perhaps we can try to use a stacking regressor to see if any model can assist LR?

Stacking Regressor

Now we try to :

- Combine Gradient Boosting, Random Forest, Decision Tree and Linear Regression using a Stacking Regressor.
- Use different stacking configurations to see which model combination performs best.

```
from sklearn.ensemble import StackingRegressor

# Base models. For gb,rf and DT, we use the optimised parameters.
rf_model = RandomForestRegressor(n_estimators=200, max_depth=10,
min_samples_split=10, random_state=42)
gb_model = GradientBoostingRegressor(n_estimators=100,
learning_rate=0.05, max_depth=3, subsample=0.7, random_state=42)
dt_model = DecisionTreeRegressor(max_depth=5, min_samples_leaf=5,
min_samples_split=2, random_state=42)
lin_model = LinearRegression()

# Stacking configurations and must include LR
stacking_models = {
    "Stacking_RF_GB_DT": StackingRegressor(
        estimators=[("rf", rf_model), ("gb", gb_model), ("dt",
dt_model)], final_estimator=LinearRegression()
```

```
    ),
    "Stacking_RF_DT_LR": StackingRegressor(
        estimators=[("rf", rf_model), ("dt", dt_model), ("lr",
lin_model)],
final_estimator=GradientBoostingRegressor(n_estimators=100,
learning_rate=0.05, random_state=42)
    ),
    "Stacking_GB_DT_LR": StackingRegressor(
        estimators=[("gb", gb_model), ("dt", dt_model), ("lr",
lin_model)], final_estimator=RandomForestRegressor(n_estimators=100,
random_state=42)
    ),
    "Stacking_RF_GB_DT_LR": StackingRegressor(
        estimators=[("rf", rf_model), ("gb", gb_model), ("dt",
dt_model), ("lr", lin_model)],
final_estimator=GradientBoostingRegressor(n_estimators=100,
learning_rate=0.05, random_state=42)
    ),
    "Stacking_RF_LR": StackingRegressor(
        estimators=[("rf", rf_model), ("lr", lin_model)],
        final_estimator=GradientBoostingRegressor(n_estimators=100,
learning_rate=0.05, random_state=42)
    ),
    "Stacking_GB_LR": StackingRegressor(
        estimators=[("gb", gb_model), ("lr", lin_model)],
        final_estimator=RandomForestRegressor(n_estimators=100,
random_state=42)
    ),
    "Stacking_DT_LR": StackingRegressor(
        estimators=[("dt", dt_model), ("lr", lin_model)],
        final_estimator=GradientBoostingRegressor(n_estimators=100,
learning_rate=0.05, random_state=42)
    )
}

# Train and evaluate each stacking model
results = {}
for name, model in stacking_models.items():
    print(f"Training {name}...")
    pipeline = Pipeline([
        ("preprocessor", full_transformer),
        ("model", model)
    ])

    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
```

```
    results[name] = {"MSE": mse, "R²": r2}

# Print results
for name, result in results.items():
    print(f"{name} - MSE: {result['MSE']:.2f}, R²: {result['R²']:.2f}\
n")
```

```
Training Stacking_RF_GB_DT...
Training Stacking_RF_DT_LR...
Training Stacking_GB_DT_LR...
Training Stacking_RF_GB_DT_LR...
Training Stacking_RF_LR...
Training Stacking_GB_LR...
Training Stacking_DT_LR...
Stacking_RF_GB_DT - MSE: 4528687.10, R²: 0.79

Stacking_RF_DT_LR - MSE: 4703000.71, R²: 0.78

Stacking_GB_DT_LR - MSE: 5485862.61, R²: 0.75

Stacking_RF_GB_DT_LR - MSE: 4751364.50, R²: 0.78

Stacking_RF_LR - MSE: 4675307.36, R²: 0.79

Stacking_GB_LR - MSE: 5392358.96, R²: 0.75

Stacking_DT_LR - MSE: 4606263.94, R²: 0.79
```

- Linear Regression seems to remain the best model based on it's R^2 value. This was despite multiple trial and errors, including letting LR be the final estimator. This resulted in all models having an R^2 value of approx 0.8. In any case, standalone LR has an R^2 score of 0.94. Hence, we decided to just simply go ahead with that model for our predictions.

```
# Create a DataFrame with actual and predicted values
results_df = X_test.copy()  # Copy test features for reference
results_df['actual_MonthlyIncome'] = y_test.values  # Add actual
target values
results_df['predicted_MonthlyIncome'] = y_pred  # Add predictions

# View or save the result
results_df.head(20)
```

```
      JobLevel  Age  DistanceFromHome  Education
NumCompaniesWorked  \
1041         2   28                 5          3                    0

184          2   53                13          2                    1

1222         1   24                22          1                    1
```

```
67            3   45                 7          3                   2

220           2   36                 5          2                   8

494           1   34                14          3                   1

430           1   35                22          3                   0

240           1   39                 1          4                   7

218           3   45                 6          3                   6

49            1   35                 8          1                   1

665           1   47                 2          4                   1

926           3   43                 4          4                   3

617           2   44                 4          3                   9

361           1   40                10          4                   3

1423          1   22                 1          2                   0

1244          1   30                 2          4                   0

1250          2   29                 1          3                   8

752           1   36                16          4                   1

271           3   47                29          4                   1

1055          4   34                15          3                   7
```

```
      TotalWorkingYears  TrainingTimesLastYear  YearsAtCompany  \
1041                  6                      4               5
184                   5                      3               4
1222                  1                      2               1
67                   25                      2               1
220                  16                      3              13
494                   8                      3               8
430                   6                      2               5
240                   7                      1               3
218                  23                      2              19
49                    1                      2               1
665                   3                      3               3
926                  23                      3              21
617                  10                      2               5
361                  10                      3               7
1423                  4                      2               3
```

```
1244                 10                        2              9
1250                 10                        5              3
752                  18                        1             17
271                  10                        2             10
1055                 16                        3             14

      YearsInCurrentRole  YearsSinceLastPromotion
YearsWithCurrManager  \
1041                  4                        1
3
184                   2                        1
3
1222                  0                        0
0
67                    0                        0
0
220                  11                        3
7
494                   2                        0
6
430                   4                        4
3
240                   2                        1
2
218                   7                       12
8
49                    0                        0
1
665                   2                        1
2
926                   7                       15
17
617                   2                        2
3
361                   7                        1
7
1423                  2                        1
2
1244                  7                        0
7
1250                  2                        0
2
752                  13                       15
14
271                   7                        9
9
1055                  8                        6
9
```

```
         BusinessTravel              Department    EducationField
Gender  \
1041      Travel_Rarely                   Sales           Medical
Male
184       Travel_Rarely  Research & Development           Medical
Female
1222      Travel_Rarely         Human Resources   Human Resources
Male
67        Travel_Rarely  Research & Development     Life Sciences
Male
220       Travel_Rarely  Research & Development     Life Sciences
Male
494       Travel_Rarely                   Sales  Technical Degree
Female
430       Travel_Rarely  Research & Development     Life Sciences
Male
240       Travel_Rarely  Research & Development           Medical
Female
218          Non-Travel                   Sales           Medical
Female
49        Travel_Rarely  Research & Development     Life Sciences
Male
665       Travel_Rarely                   Sales     Life Sciences
Female
926       Travel_Rarely                   Sales         Marketing
Female
617       Travel_Rarely  Research & Development           Medical
Male
361       Travel_Rarely  Research & Development     Life Sciences
Female
1423      Travel_Rarely  Research & Development     Life Sciences
Male
1244  Travel_Frequently  Research & Development  Technical Degree
Female
1250  Travel_Frequently  Research & Development     Life Sciences
Male
752       Travel_Rarely  Research & Development     Life Sciences
Female
271          Non-Travel  Research & Development     Life Sciences
Male
1055  Travel_Frequently  Research & Development           Medical
Male

     OverTime                   JobRole MaritalStatus
actual_MonthlyIncome  \
1041       No           Sales Executive        Single
8463
184        No    Manufacturing Director      Divorced
4450
```

```
1222         No             Human Resources       Married
1555
67           No          Research Scientist      Divorced
9724
220          No       Laboratory Technician        Single
5914
494         Yes        Sales Representative      Divorced
2579
430          No       Laboratory Technician        Single
4230
240          No       Laboratory Technician      Divorced
2232
218          No             Sales Executive        Single
8865
49           No       Laboratory Technician       Married
2269
665         Yes        Sales Representative        Single
3294
926          No             Sales Executive        Single
10231
617          No  Healthcare Representative        Single
5933
361         Yes       Laboratory Technician       Married
2213
1423         No          Research Scientist        Single
3375
1244         No          Research Scientist        Single
4968
1250        Yes  Healthcare Representative        Single
6294
752          No       Laboratory Technician        Single
2743
271         Yes                     Manager       Married
11849
1055         No           Research Director      Divorced
17007

      predicted_MonthlyIncome
1041              7566.721853
184               7127.297313
1222              3927.388603
67                3118.635256
220               3118.635256
494               3118.635256
430               3118.635256
240               3118.635256
218               6444.223080
49                3118.635256
665               3132.513215
```

```
926             6491.975111
617             6240.055262
361             3118.635256
1423            3118.635256
1244            2971.171862
1250            6863.662739
752             3118.635256
271            16365.336705
1055           15067.341724
```

Feature Encoding & Train-Test Split

We also predicted **MonthlyIncome** using another set of predictors, for example, Age, DistanceFromHome, and OverTime. Then, we apply one-hot encoding to the categorical variables.

```
# Define our target variable and features
target = "MonthlyIncome"
features = ["Age", "DistanceFromHome", "OverTime", "BusinessTravel",
"JobLevel"]

# One-hot encode the categorical columns among the features
df_encoded = pd.get_dummies(df[features], drop_first=True)

# Prepare feature matrix X and target vector y
X = df_encoded.values
y = df[target].values

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

print("Training set shape:", X_train.shape)
print("Test set shape:", X_test.shape)

Training set shape: (1176, 6)
Test set shape: (294, 6)
```

Train a Random Forest Regressor

We also trained a Random Forest Regressor on our training data. This model is robust, handles non-linearities well, and can provide feature importance insights.

```
# Initialize and train the Random Forest Regressor
rf_regressor = RandomForestRegressor(n_estimators=100,
random_state=42)
rf_regressor.fit(X_train, y_train)

RandomForestRegressor(random_state=42)
```

Evaluate the Regression Model

We now predict MonthlyIncome on the test set and evaluate our model using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and R^2 score.

```
# Predict MonthlyIncome on the test set
y_pred = rf_regressor.predict(X_test)

# Calculate evaluation metrics
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"Mean Absolute Error: {mae:.2f}")
print(f"Mean Squared Error: {mse:.2f}")
print(f"R^2 Score: {r2:.3f}")
```

```
Mean Absolute Error: 1100.05
Mean Squared Error: 2226383.00
R^2 Score: 0.898
```

Sample Predictions

Here, we select a few samples from the test set and compare the predicted MonthlyIncome against the actual values.

```
# Select 5 random samples from the test set
sample_indices = np.random.choice(X_test.shape[0], size=5,
replace=False)
sample_features = X_test[sample_indices]
sample_true = y_test[sample_indices]

# Predict using the model for these samples
sample_preds = rf_regressor.predict(sample_features)

# Create a DataFrame to display sample predictions
sample_df = pd.DataFrame(sample_features, columns=df_encoded.columns)
sample_df["TrueMonthlyIncome"] = sample_true
sample_df["PredictedMonthlyIncome"] = sample_preds
sample_df
```

```
  Age DistanceFromHome JobLevel OverTime_Yes
BusinessTravel_Travel_Frequently  \
0  28                8        1        False
False
1  28                1        1         True
True
2  44                4        5        False
False
```

```
3  58                  21         4          True
False
4  34                   9         3         False
False

  BusinessTravel_Travel_Rarely  TrueMonthlyIncome
PredictedMonthlyIncome
0                         True               3310
4215.476167
1                        False               2154
2821.833381
2                         True              19190
18899.860000
3                         True              17875
14858.790000
4                         True               8500
9434.635000
```