Predict salary_usd

Analyze ai_job_Salary

Version	Author	Date
1.0	admin no email	2025-07-28 09:01:26

Table of contents

I. Executive Summary

A Regression Machine Learning model was built using Dataiku DSS Visual ML. Its goal is to predict **salary_usd** given a total of 31 features. Using a dataset of 11944 rows, the process led to the selection of the Random Forest algorithm.

A. Methodology

To ensure a good generalization capability for the ML model, a test strategy was set up. Data on which ML candidate models were not trained on was used for this purpose. The testing strategy was the following:

Policy	Split the dataset
Use time ordering	No
Sampling method	No sampling (whole data)
Split mode	Random
Use K-fold cross-testing	No
Train ratio	0.8
Random seed	1337

See section II.D for detailed explanations about these options.

Before being tested, the ML candidate models had been tuned to find the best combination of hyperparameters according to the R2 Score metric. This optimal hyperparameter search, based on assessing performance on a validation set, was done using the following methodology:

Search strategy	
Strategy	Grid search
Search parameters	
Randomize grid search	Yes
Random state (hyperparameter search)	1337
Max number of iterations	0 (no limit)

Max search time	0 (no limit)
Parallelism	4
Cross-validation	
Cross-validation strategy	K-fold
Number of folds	5
Random state (cross-validation split)	1337
Grouped	No

See section **Erreur ! Source du renvoi introuvable.** for detailed explanations about these options.

B. Results

The Random Forest algorithm was selected. The evaluation metric used to tune the hyperparameters was R2 Score computed on the validation dataset. After the best hyperparameter combination was found, the same metric was also computed on the test dataset. The final value was 0.893.

II. Methodology

This section deals with the methodological details:

- The *Problem Definition* consists of selecting the target (**salary_usd**) and the type of problem (Regression).
- *Data Ingestion* analyzes each feature in order to maximize its prediction potential.
- *Model and Feature Tuning* describes the tested algorithms and the way to find the best hyperparameter set for each of them.
- The *Model Evaluation and Selection* strategy indicates how to compute the metrics that allow for comparison between the best-tuned algorithms so that the user can select the best algorithm according to one of the computed metrics (Here Random Forest).

A. Problem Definition

A Regression Machine-Learning model was built using Dataiku DSS. Its goal is to predict **salary_usd** given a total of 31 features.

B. Data Ingestion

During the data ingestion phase, the features are transformed into numerical features without missing values so as to be ingestible by the Machine Learning algorithm. The table below summarizes the processing applied to each of them.

Feature Name	Status	Type	Processing
employment_type	Input	Category	Dummy encoding
has_linux	Input	Numeric	Avg-std rescaling
years_experience	Input	Numeric	Avg-std rescaling
has_spark	Input	Numeric	Avg-std rescaling
has_R	Input	Numeric	Avg-std rescaling
industry	Input	Category	Dummy encoding
has_tensorflow	Input	Numeric	Avg-std rescaling
has_azure	Input	Numeric	Avg-std rescaling
salary_usd	Target	Numeric	
has_mlops	Input	Numeric	Avg-std rescaling
required_skills	Rejected	Text	
has_dataviz	Input	Numeric	Avg-std rescaling
has_aws	Input	Numeric	Avg-std rescaling
job_title	Input	Category	Dummy encoding
benefits_score	Input	Numeric	Avg-std rescaling
has_gcp	Input	Numeric	Avg-std rescaling
has_git	Input	Numeric	Avg-std rescaling
has_sql	Input	Numeric	Avg-std rescaling
remote_ratio	Input	Numeric	Avg-std rescaling

has_tableau	Input	Numeric	Avg-std rescaling
has_deep_learning	Input	Numeric	Avg-std rescaling
job_description_length	Input	Numeric	Avg-std rescaling
has_statistics	Input	Numeric	Avg-std rescaling
has_docker	Input	Numeric	Avg-std rescaling
has_hadoop	Input	Numeric	Avg-std rescaling
education_required	Input	Category	Dummy encoding
company_location	Input	Category	Dummy encoding
has_pytorch	Input	Numeric	Avg-std rescaling
experience_level	Input	Category	Dummy encoding
company_size	Input	Category	Dummy encoding
has_Python	Input	Numeric	Avg-std rescaling

Legend

- *Feature name:* Name of the feature column
- *Feature status:* Input, Target or Rejected
- *Feature type:* Numeric, Category, Text, or Array
- *Processing:* Type of processing applied (Avg-std rescaling, dummy-encode…)

C. Model and Feature Tuning

1. Pre-processings

Once each feature has been processed, it is possible to combine them to generate new features:

- Pairwise linear feature generation: Disabled
- Pairwise polynomial feature generation (A*B) for all pairs of features: Disabled

2. Tested Algorithms

A selection of algorithms (candidate models) was then trained on the Machine Learning dataset, with various combinations of hyperparameters. The section below details the tested

algorithms and the space of hyperparameters for each of them. It begins with the selected algorithm and its hyperparameter selection and continues with the other tested algorithms.

a) Selected Model

The Random Forest algorithm has been finally selected.

A Random Forest is made of many decision trees. Each tree in the forest predicts a record, and each tree "votes" for the final answer of the forest.
The forest chooses the class having the most votes.
A decision tree is a simple algorithm which builds a decision tree. Each node of the decision tree includes a condition on one of the input features.
When "growing" (ie, training) the forest:
- for each tree, a random sample of the training set is used;
- for each decision point in the tree, a random subset of the input features is considered.
Random Forests generally provide good results, at the expense of "explainability" of the model.

The settings for this algorithm are given below. For hyperparameters, the possible values or ranges are listed:

Number of trees	100
Feature sampling strategy	Fixed proportion
Number of features to sample	5
Proportion of features to sample	1
Maximum depth of tree	6 15
Minimum samples per leaf	10
Parallelism	1
Allow sparse matrices	Yes

b) Alternative Models

Other algorithms are also tested. They are listed below, along with their settings:

(1) Ridge Regression

Ridge Regression is a linear model that addresses some problems of Ordinary Least Squares by imposing a penalty (or regularization term) to the weights
Ridge regression uses a L2 regularization

Regularization term	Specify values to test
Alpha	0.1 1 3

(2) XGBoost

XGBoost is an advanced gradient tree boosting algorithm. It has support for parallel processing, regularization and early stopping, which makes it a fast, scalable and accurate algorithm.
For more information on gradient tree boosting, see the "Gradient tree boosting" algorithm.

Booster	Try Gradient Boosted Trees: Yes Try DART: No
Objective	Try Regression with squared loss: Yes Try Logistic Regression: No Try Gamma Regression: No Try Poisson Regression: No Try Tweedie Regression: No Use 'Logistic Regression' only if the target is a probability (soft target) with values in [0, 1]. Useful for any outcome that might be Tweedie-distributed, like total loss in insurance.
Tree method	Automatic - CPU only
Maximum number of trees	300
Early stopping	Yes
Early stopping rounds	4
Maximum depth of tree	3
Learning rate	0.2

Max delta step	0
L2 regularization	1
L1 regularization	0
Gamma	0
Minimum child weight	1
Subsample	1
Columns subsample ratio for trees	1
Columns subsample ratio for splits / levels	1
Parallelism	4
Allow sparse matrices	Yes
Custom missing value	No

3. Hyperparameter Search

The hyperparameter search is done for each algorithm separately. It consists of finding the combination of hyperparameters that results in the best-trained model according to the validation metric (R2 Score) computed on the validation dataset.
The actual search settings for all the tested algorithms, including the selected one, are the following:

Search strategy	
Strategy	Grid search
Search parameters	
Randomize grid search	Yes
Random state (hyperparameter search)	1337
Max number of iterations	0 (no limit)
Max search time	0 (no limit)
Parallelism	4
Cross-validation	
Cross-validation strategy	K-fold
Number of folds	5

Random state (cross-validation split)	1337
Grouped	No

Legend

- *Randomize grid search:* If true, the grid was shuffled before the search.
- *Max number of iterations:* This parameter sets the number of points of the grid that have been evaluated.
- *Max search time:* Maximum search time in minutes.
- *Parallelism:* -1 for automatic. It sets the number of hyperparameter searches that are performed simultaneously.
- *Stratified:* If true, the same target distribution is kept in all the splits.

Illustration:

Note: A grey area appears on the graphic to illustrate the data that is used for the test dataset.

4. ML Overrides

Model overrides ensure predetermined prediction outcomes based on a set of defined rules.

ML Overrides
No overrides defined in the settings

D. Evaluation and Selection

The last part of the methodology consists of comparing the performance of each algorithm trained using the best hyperparameter combination. The policy can consist in either:

- Splitting the dataset by setting apart a test dataset, also called the hold-out dataset, for this performance evaluation. The train ratio indicates the amount of the dataset used in training, the remaining being used for evaluation.
- Performing a K-fold evaluation. It allows a more precise performance evaluation, at the expense of increased computation time.

This is indicated by the policy and the split mode in the table below.

When the original dataset is very big, the required computational resources may be too large compared to the expected benefit of training algorithms on it. As a result, the training, validation, and testing may be performed on a subset of the dataset. The sampling method given in the table below defines how it is built.

Policy	Split the dataset
Use time ordering	No
Sampling method	No sampling (whole data)
Split mode	Random
Use K-fold cross-testing	No
Train ratio	0.8
Random seed	1337

Illustration:

No sampling (whole data) & 0.8 train ratio

Sampling 

Evaluation 

The metrics used to rank models obtained by different algorithms are computed on the test set. The final model is trained on the train set.

Legend

- Policy:
  - *Split the dataset:* Split a subset of the dataset.
  - *Explicit extracts from the dataset:* Use two extracts from the dataset, one for the train set, one for the test set.
  - *Explicit extracts from two datasets:* Use two extracts from two different datasets, one for the train set, one for the test set.
  - *Split another dataset:* Split a subset of another dataset, compatible with the dataset.
  - *Explicit extracts from another dataset:* Use two extracts from another dataset, one for the train set, one for the test set.
- *Sampling method:* A subset may have been extracted in order to limit the computational resources required by the evaluation and selection process. The *Record limit* gives its size.
  - *No sampling (whole data)*: the complete dataset has been kept.
  - *First records*: The first N rows of the dataset have been kept (or all the dataset if it has fewer rows. The current dataset has 11944 rows). It may result in a very biased view of the dataset.
  - *Random (approx. ratio)*: Randomly selects approximately X% of the rows.
  - *Random (approx. nb. records)*: Randomly selects approximately N rows.
  - *Column values subset (approx. nb. records)*: Randomly selects a subset of values and chooses all rows with these values, in order to obtain approximately N rows. This is useful for selecting a subset of customers, for example.
  - *Class rebalance (approx. nb. records)*: Randomly selects approximately N rows, trying to rebalance equally all modalities of a column. It does not

oversample, only undersamples (so some rare modalities may remain under-represented). Rebalancing is not exact.
    - *Class rebalance (approx. ratio)*: Randomly selects approximately X% of the rows, trying to rebalance equally all modalities of a column. It does not oversample, only undersamples (so some rare modalities may remain under-represented). Rebalancing is not exact.
- Partitions:
    - *All partitions:* Use all partitions of the dataset.
    - *Select partitions:* Use an explicitly selected list of partitions.
    - *Latest partition:* Use the "latest" partition currently available in the dataset. "Latest" is only defined for single-dimension time-based partitioning.
- *Time variable:* By enabling time-based ordering, DSS checks that the train and the test sets are consistent with the time variable. Moreover, DSS guarantees that:
    - The train set is sorted according to the selected variable.
    - The hyperparameter search is done with training sets and validation sets consistent with the ordering induced by the time variable.
- *Split mode:* If "*K-fold cross-test*" is selected, it gives error margins on metrics, but strongly increases training time.
- *Train ratio:* Proportion of the sample that goes to the train set. The rest goes to the test set.
- *Number of folds:* Number of folds K to divide the dataset into.
- *Random seed:* Using a fixed random seed allows for reproducible results.

III. Experiment Results

The methodology detailed in the previous section has been run. The obtained results are presented in this section.

A. Selected Model

Random Forest was finally selected by the user with the optimal set of hyperparameters given in the table below:

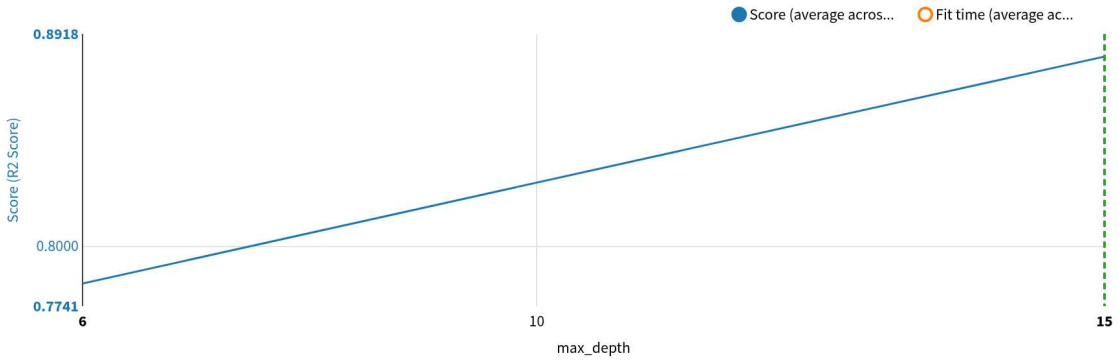
Algorithm	Random forest regression	Split quality criterion	MSE

Number of trees	100	Use bootstrap	Yes
Max trees depth	15	Feature sampling strategy	prop
Min samples per leaf	10	Used features	100%
Min samples to split	30		

See section II.C.2.a) for detailed explanations on the algorithm and its hyperparameters.

B. Alternative Models

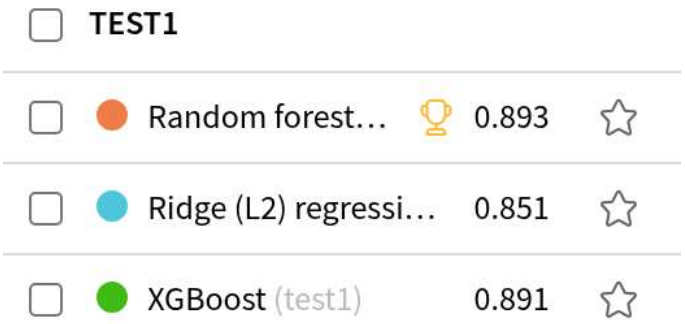
For the selected algorithm, the following other hyperparameter combinations were tried and led to lower performance. As an example, the plot below shows the evolution of the performance for each hyperparameter:



The table below lists all the performed trainings:

max_-depth	Score	Score StdDev	Fit Time	Fit Time StdDev	Score Time	Score Time StdDev
6	0.783899	0.008713	29.580200	0.683583	0.646600	0.128550
15	0.881977	0.004910	75.255400	21.273568	0.551600	0.272675

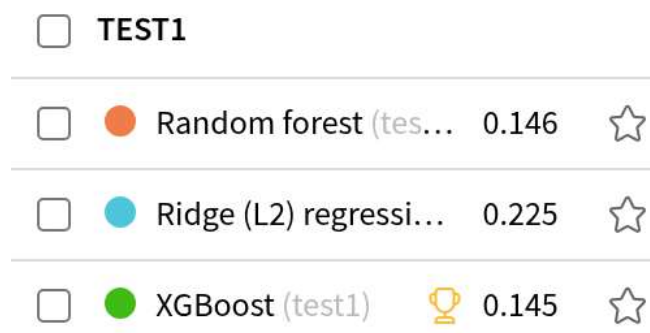
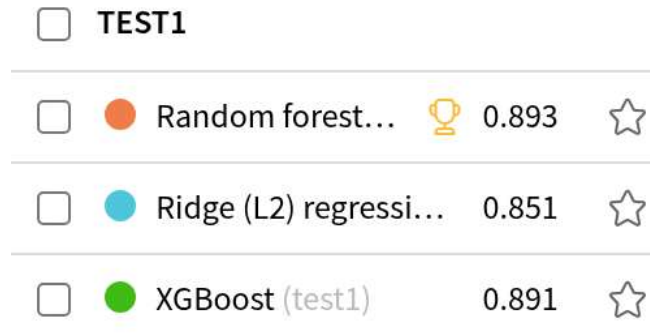
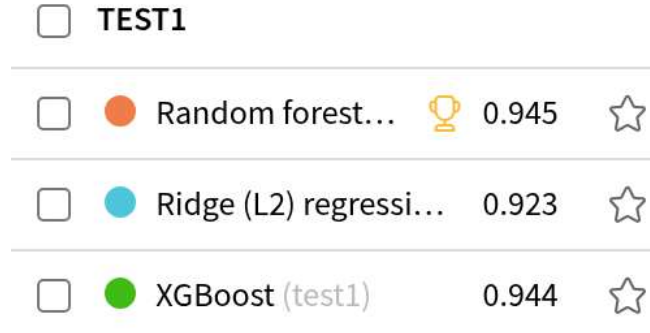
The selected algorithm was compared with other algorithms. The table below gives the performance obtained with the combination of hyperparameters that optimizes the R2 Score metric:



Complete performance results obtained with the other evaluated metrics are given below:

EVS	
MAPE	

MAE	 TEST1 Random for… 1.45e+4 Ridge (L2) regres… 1.73e+4 XGBoost (test1) 1.45e+4
MSE	 TEST1 Random for… 4.02e+8 Ridge (L2) regres… 5.62e+8 XGBoost (test1) 4.13e+8
RMSE	 TEST1 Random for… 2.01e+4 Ridge (L2) regres… 2.37e+4 XGBoost (test1) 2.03e+4

RMSLE	
R2 score	
Pearson coeff.	

IV. Selected Model Results

A. Selected Model Metrics

The detailed metrics obtained on the test dataset are given below.

Explained Variance Score	Best possible score is 1.0, lower values are worse	0.8935
Mean Absolute Percentage Error (MAPE)	Average of the absolute value of the relative regression error	12.42%
Mean Absolute Error (MAE)	Average of the absolute value of the regression error	1.446e+4
Mean Squared Error (MSE)	Average of the squares of the errors	4.023e+8
Root Mean Squared Error (RMSE)	Square root of the MSE	2.006e+4
Root Mean Squared Logarithmic Error (RMSLE)	Root of the average of the squares of the natural log of the regression error	0.1457
R2 Score	(Coefficient of determination) regression score function	0.8935
Pearson Coefficient	Correlation coefficient between actual and predicted values. +1 = perfect correlation, 0 = no correlation, -1 = perfect anti-correlation	0.9452

The ml assertions metrics are given below.

Name	Criteria	Expected range	Expected valid ratio	Rows matching criteria	Rows dropped by the model	Valid ratio	Result
	No assertions defined in the settings						

The ml override metrics are given below.

ML Overrides

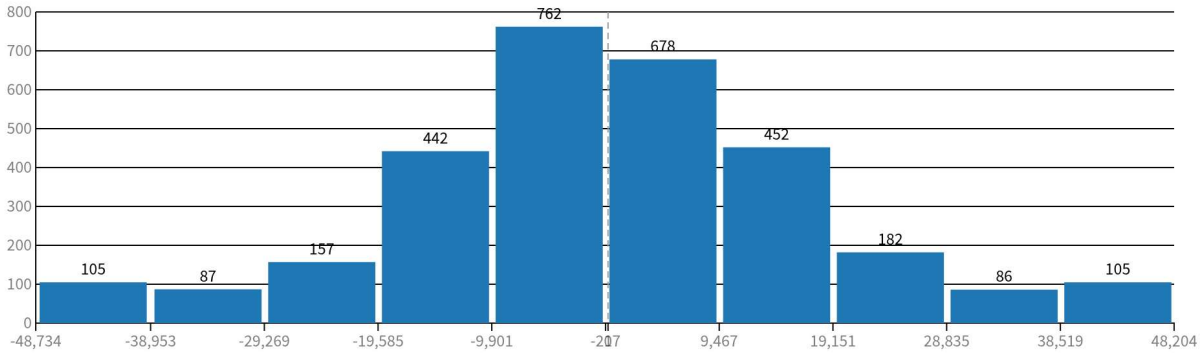
No overrides defined in the settings

B. Selected Model Performance Charts

The error distribution table for this regression model is given below as a table with some statistics, as well as a histogram and as a scatter plot.

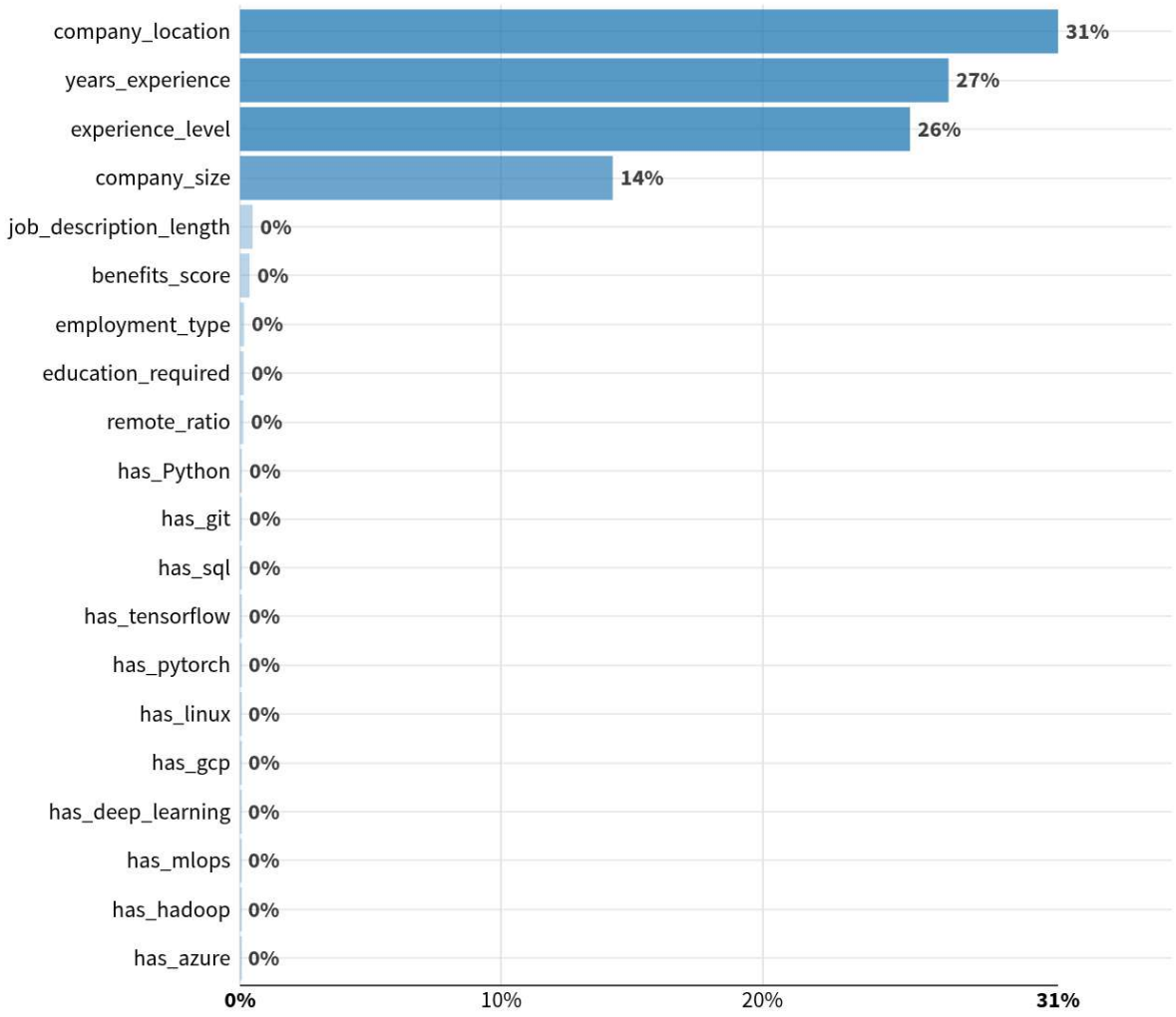
Min.	-88736
2nd perc.	-48637
25th perc.	-10374
Median	-577.88
75th perc.	10304
90th perc.	21433
98th perc.	48204
Max.	80453
Average	-147.91
Average (clipped 2nd-98th perc.)	-122.22
Standard deviation	20060
Standard deviation (clipped 2nd-98th perc.)	18617

The errors (the difference between predicted and actual values) should be centered around zero, and the distribution should be “narrow”, i.e., the spread of the error should be limited. More generally, the errors should be “normally” distributed around zero (the curve should look like a bell).
To reduce the effect of possible spurious outliers, the error distribution is winsorized (clipped) at the 2nd and 98th percentiles.

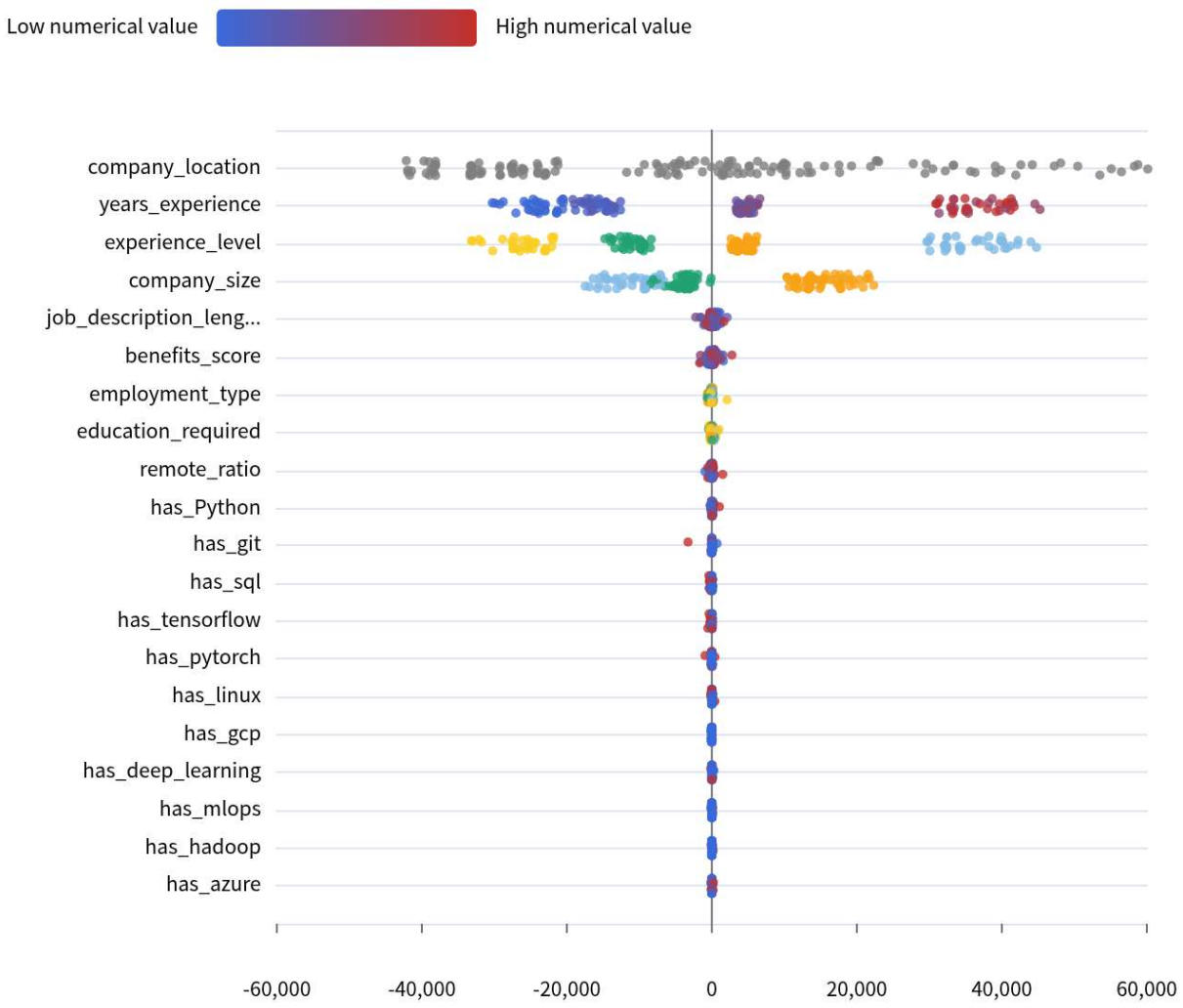


C. Sensitivity Testing and Analysis

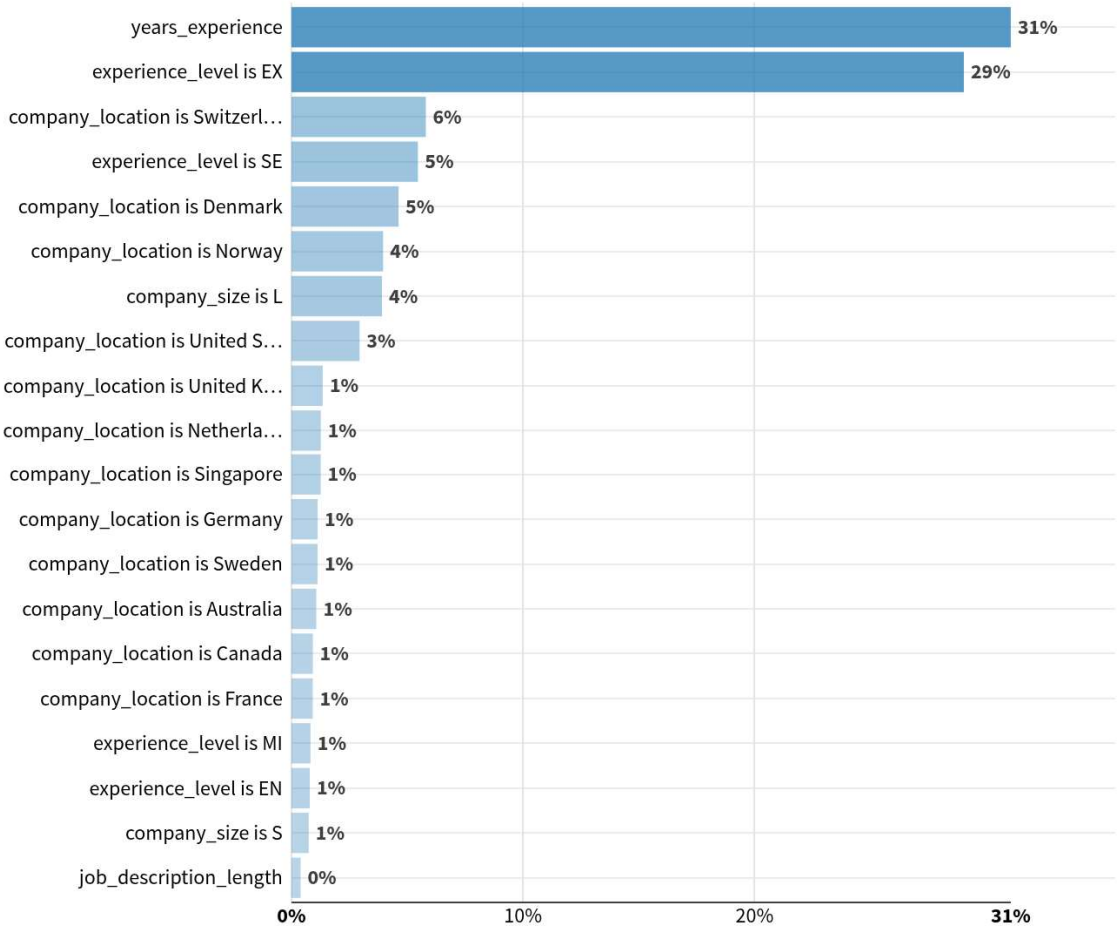
Shapley feature importance has been computed representing which features have the strongest impact on the predictions of the algorithm.



Feature effects display multiple Shapley values computed per feature.



The selected algorithm has provided Gini feature importance values that assess which features have a significant impact on its performance.



D. Diagnostics

ML Diagnostics are designed to identify and help troubleshoot potential problems and suggest possible improvements at different stages of training and building machine learning models.

Dataset sanity checks	Nothing to report
Modeling parameters	Nothing to report
Training speed	Nothing to report
Training reproducibility	Nothing to report
Overfit detection	Nothing to report

Leakage detection	Nothing to report
Model check	Nothing to report
ML assertions	Nothing to report
Abnormal predictions detection	Nothing to report
Scoring dataset sanity checks	Nothing to report
Evaluation dataset sanity checks	Nothing to report
Time series resampling checks	Nothing to report
Treatment checks	Nothing to report
Propensity checks	Nothing to report
Evaluation error	Nothing to report

V. Deployment and Monitoring

A. Implementation Details

- The backend used by the model is: Python (in memory)
- The model can be found here: https://dss-82c4dc21-8ca5fa5e-dku.eu-west-3.app.dataiku.io/projects/PROJET_EMPLOI/analysis/CgWkvfl0/ml/p/RDmJzm7w/A-PROJET_EMPLOI-CgWkvfl0-RDmJzm7w-s1-pp1-m1/report
- The name of the generated file is: Dataiku Model Documentation - Predict salary_usd on Analyze ai_job_Salary - Random Forest.docx
- The timing of the training was the following:

Preprocessed in	0.9s
Trained in	196.1s
Loading train set	0.3s
Loading test set	0.1s
Collecting statistics	0.1s
Preprocessing train set	0.3s
Preprocessing test set	0.1s

Hyperparameter searching	149.6s
Fitting model	29.1s
Saving model	0.9s
Scoring model	1.2s
Preparing explainability	15.3s

B. Version Control

- The model was trained at 2025-07-25 14:22:56 (In the DSS server time zone).
- The model was trained with the following version of DSS: 14.0.0
- With the following code environment: DSS builtin environment

VI. Annexes

The first 3 levels of the decision tree are represented below: