**Manpower Research and Statistics Department – Data Analytics Assessment**

Note: This is a 2-part assessment.

You may use the following programming language (Python or R studio) and any data visualisation tools of your preference for this assessment.

Submit a report in word document format containing snippets of the relevant codes and outputs (i.e. tables/plots) with short descriptions of each process and insights drawn from your analysis/prediction. Please also include any assumptions or hypothesis which you have made.

**Tasks**

With an interest area in mind, generate/perform following:

Data Inspection

Data Cleaning

Descriptive Statistics

Descriptive plots

Data Analysis/Predictive analytics

Any other additional technical skills which you may want to include.

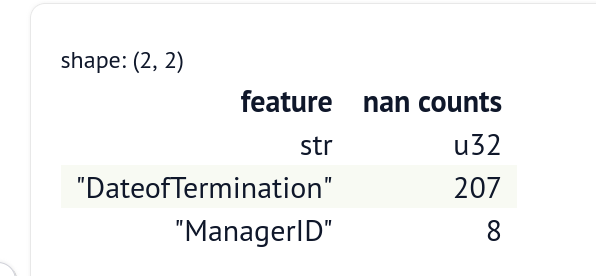
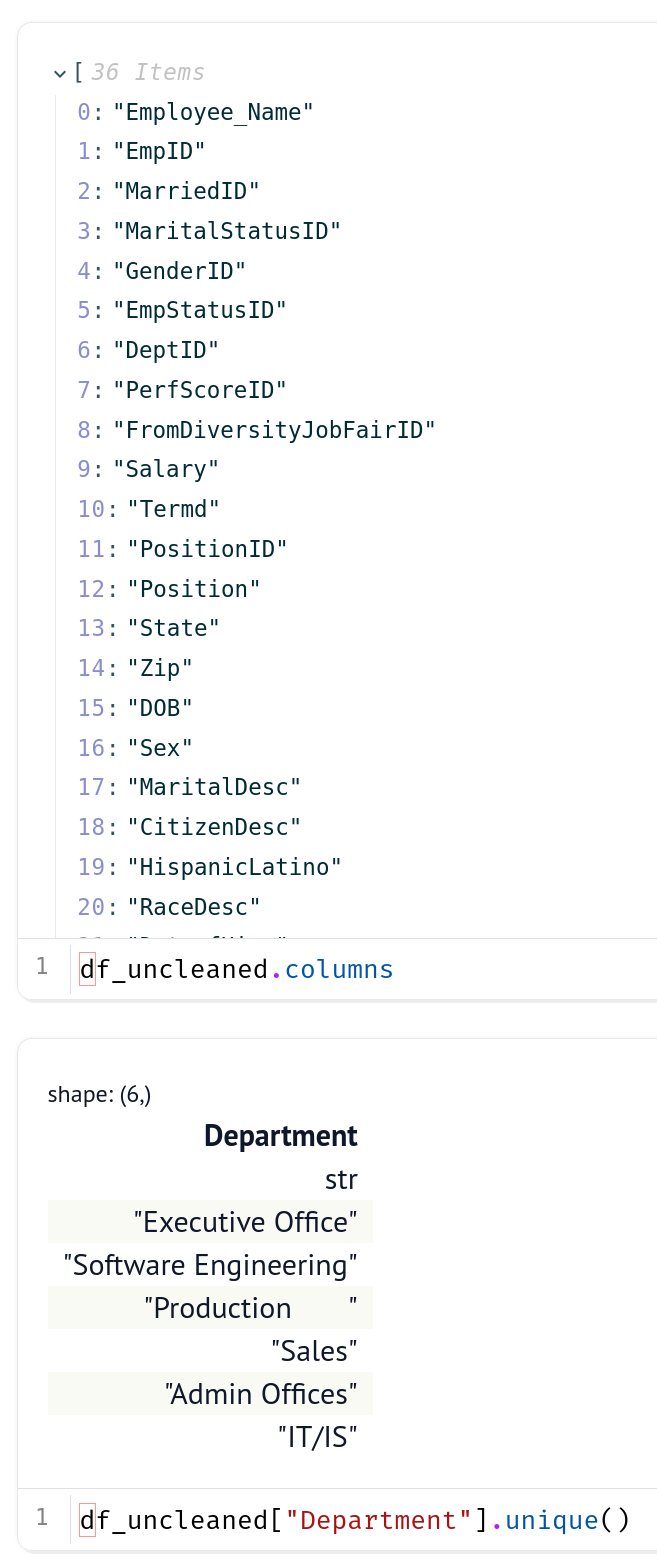
With the introduction of Artificial Intelligence and Large Language Model how do you think the roles of data analyst and data scientist will be affected?

Dataset selected: https://www.kaggle.com/datasets/rhuebner/human-resources-data-set

## There is alot of insights to go about here but I would like to know employees satisfaction and engagement patterns

From the pictures below I there are things I wanted to clean:

- department string values spacing  
- standardising naming convention in feature names. Verbosity in column names has increased for clarity

-nan replacements

Codes used for cleaning:

df = pl.read\_csv("HRDataset\_v14.csv")

df.columns = [

"EmployeeName",

"EmployeeID",

"MarriedID",

"MaritalStatusID",

"GenderID",

"EmpStatusID",

"DepartmentID",

….. and so one

]

df = df.with\_columns(

pl.col(

"DateOfHire",

)

.str.to\_date("%m/%d/%Y")

.cast(pl.UInt64),

pl.col("DateOfBirth").str.to\_date("%m/%d/%y").cast(pl.UInt64),

pl.col("DateOfTermination").str.to\_date("%m/%d/%Y").cast(pl.UInt64),

pl.col("LastPerformanceReviewDate")

.str.to\_date("%m/%d/%Y")

.cast(pl.UInt64),

pl.col("Department").str.strip\_chars(),

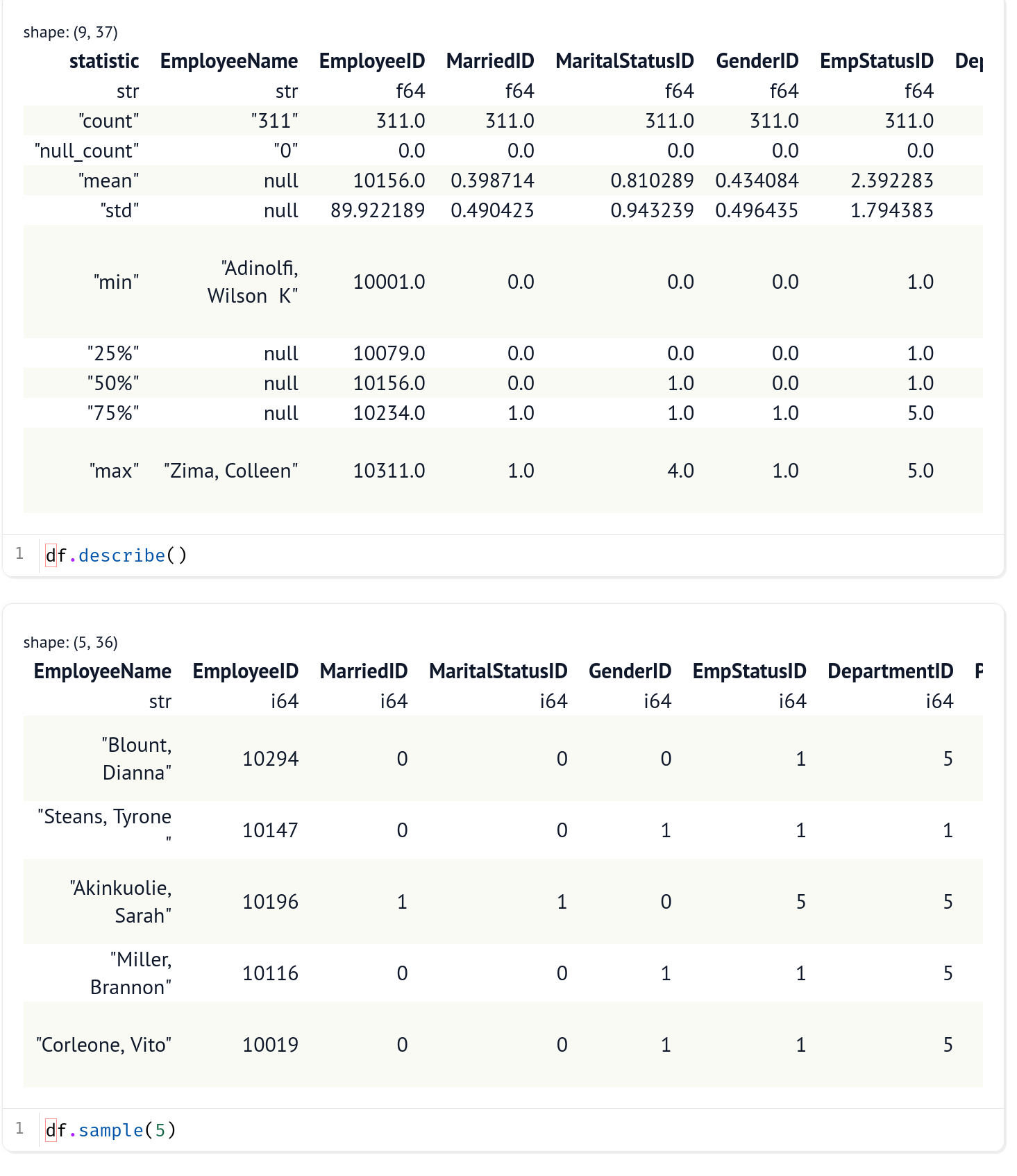
pl.col("ManagerID").fill\_null(-1),

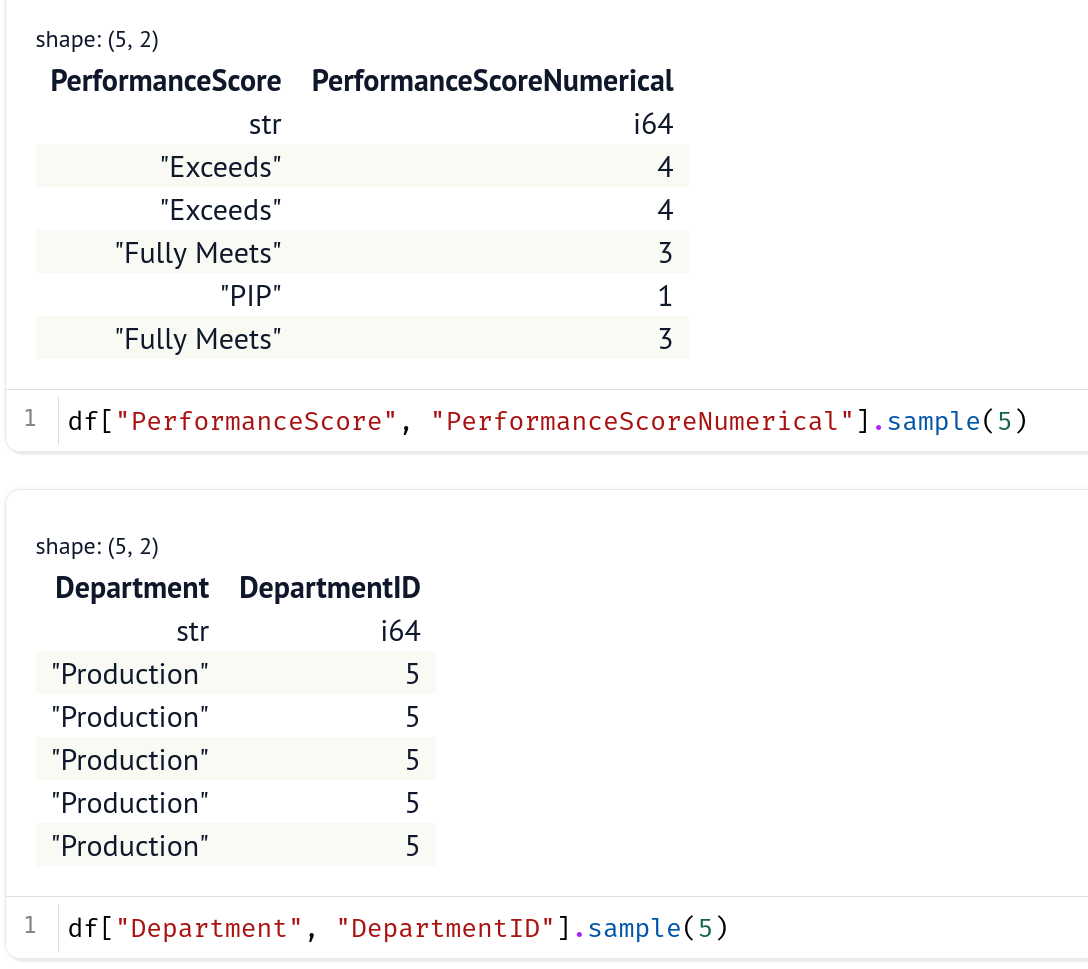
).with\_columns(

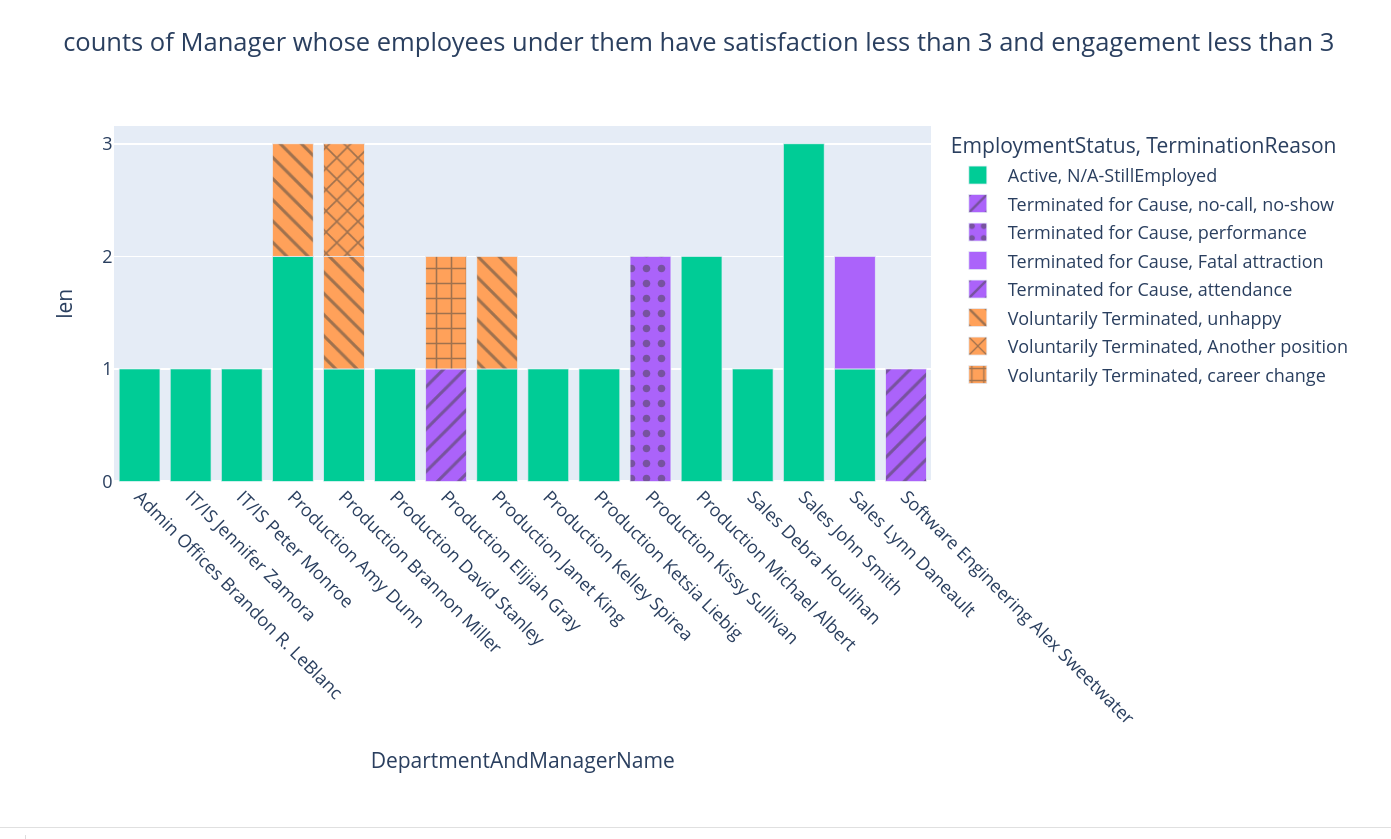
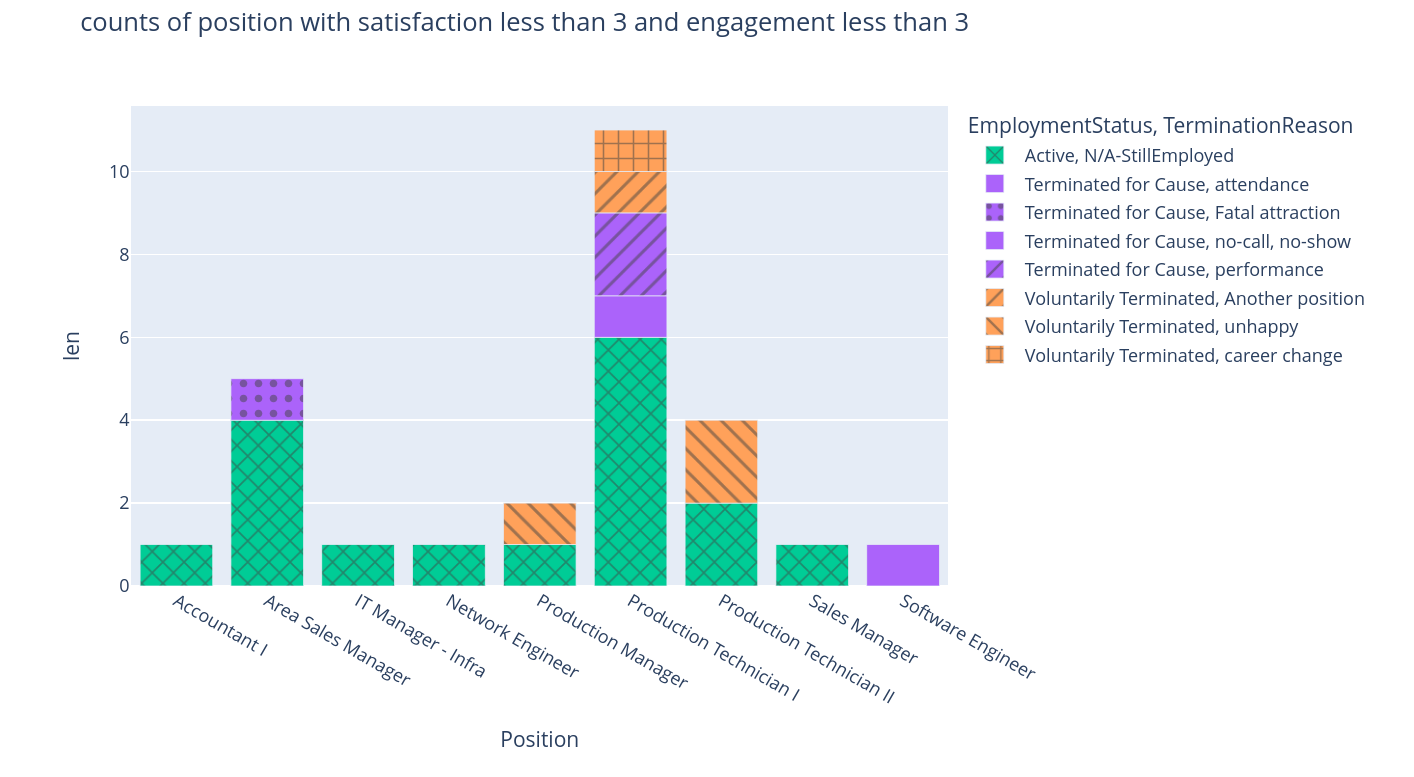
pl.col("DateOfTermination").fill\_null(0),

)

Used Polars describe() and sample() to get a sense of how the data is like:



**There are correspondence between ID suffixed features that is of int type and the feature that is of string type. Here are some examples below:**



**Production, software engineering, sales department has the highest proportion of poor performers under the score of 3.**

**Production, software engineering, sales departments are the only departments who have satisfaction and engagement less than the score of 3**

**production has some volunteery resignations because of unhappiness**

Legends below are of feature name to the number that is shown on the scatterplot matrix below. ScatterPlot Matrix from plotly is faster than Seaborn pairplot but I can't rotate the full feature name for some reason.

EmployeeID = EmpID

EmpStatusID = EmpStatusID

DepartmentID = DepartmentID

PerformanceScoreNumerical = PerfScoreID

FromDiversityJobFairID = DEIJobFair

Salary = Salary

Termd = Termd

PositionID = PositionID

ManagerID = ManagerID

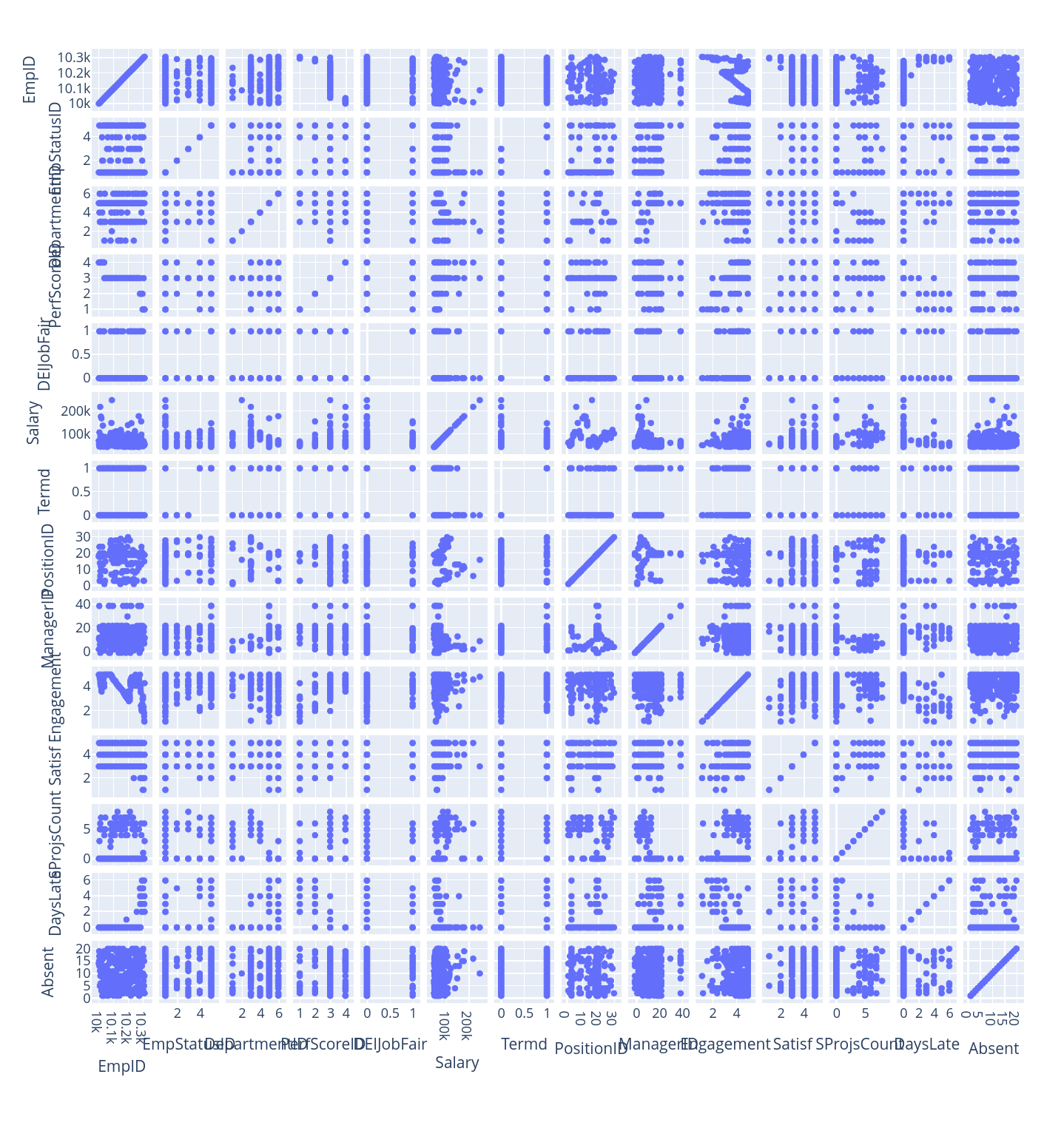
EngagementSurvey = Engagement

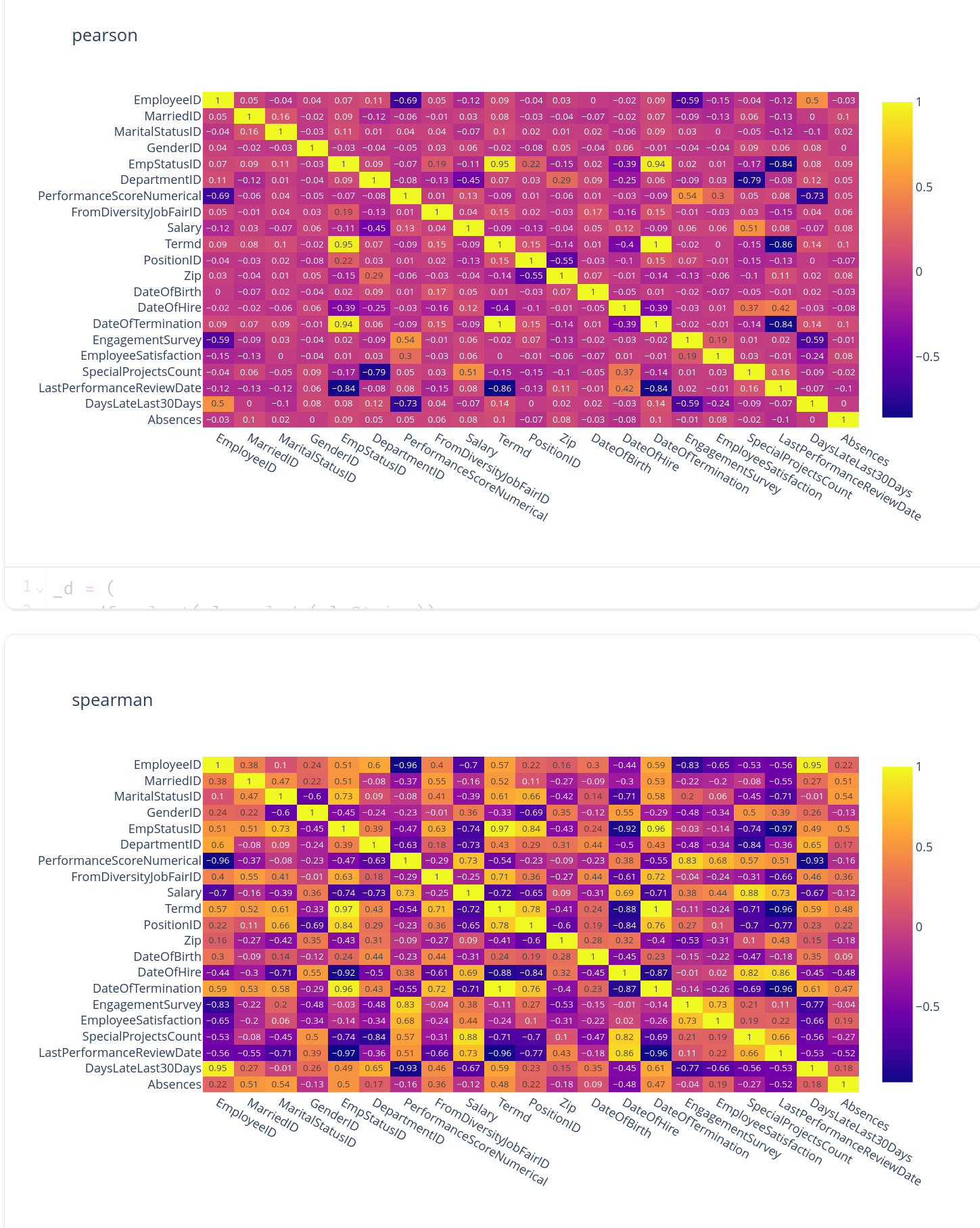
EmployeeSatisfaction = Satisf

SpecialProjectsCount = SProjsCount

DaysLateLast30Days = DaysLate

Absences = Absen

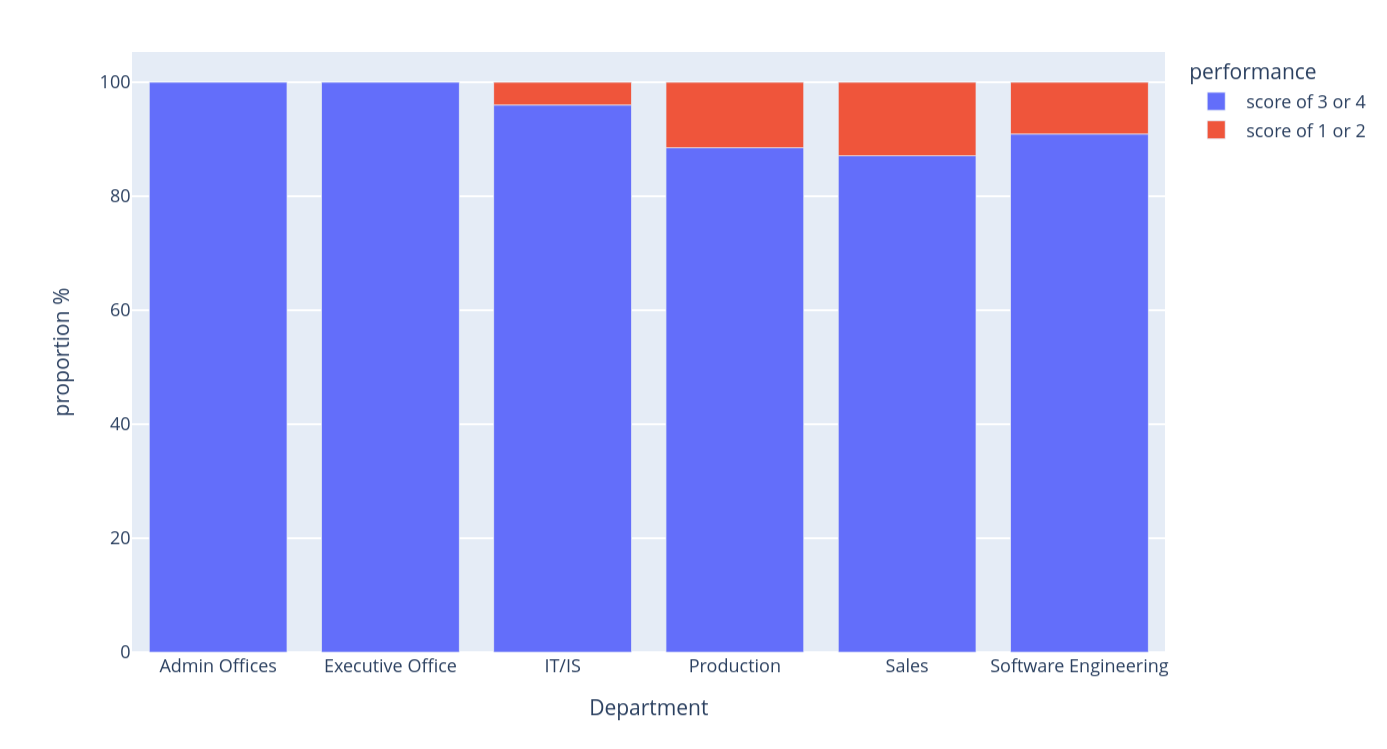


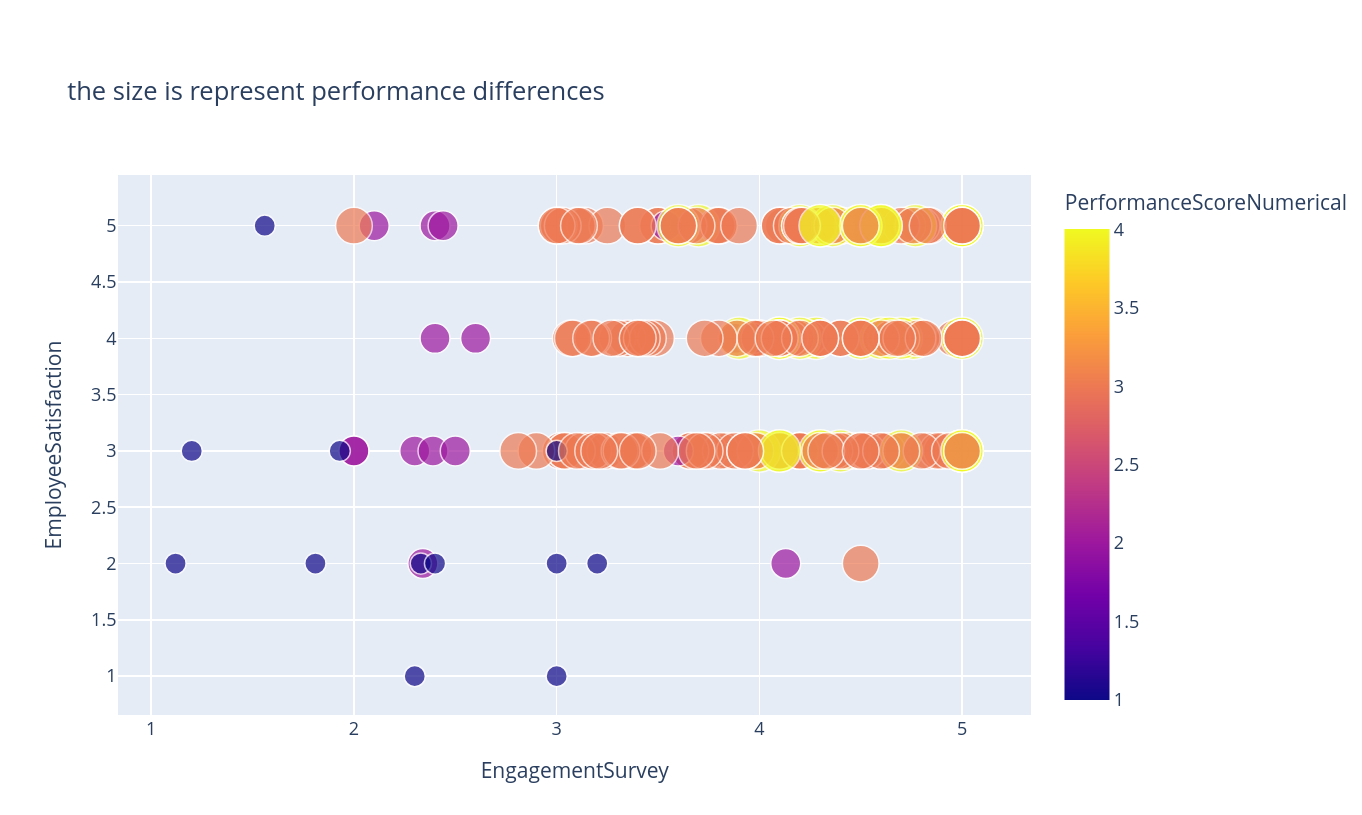


correlations of my interests :

* performance score id | dayslatelast30
* departmentid | special projects count
* engagementStudy | PerformanceScoreNumerical
* EmployeeSatisfaction | PerformanceScoreNumerical
* Salary | special projects count
* engagementStudy | dayslatelast30

**I used Spearman too just in case of the linear correlations comparision of person is misleading. Although to be fair the scatter plot doesn’t show anything very obvious linearity**

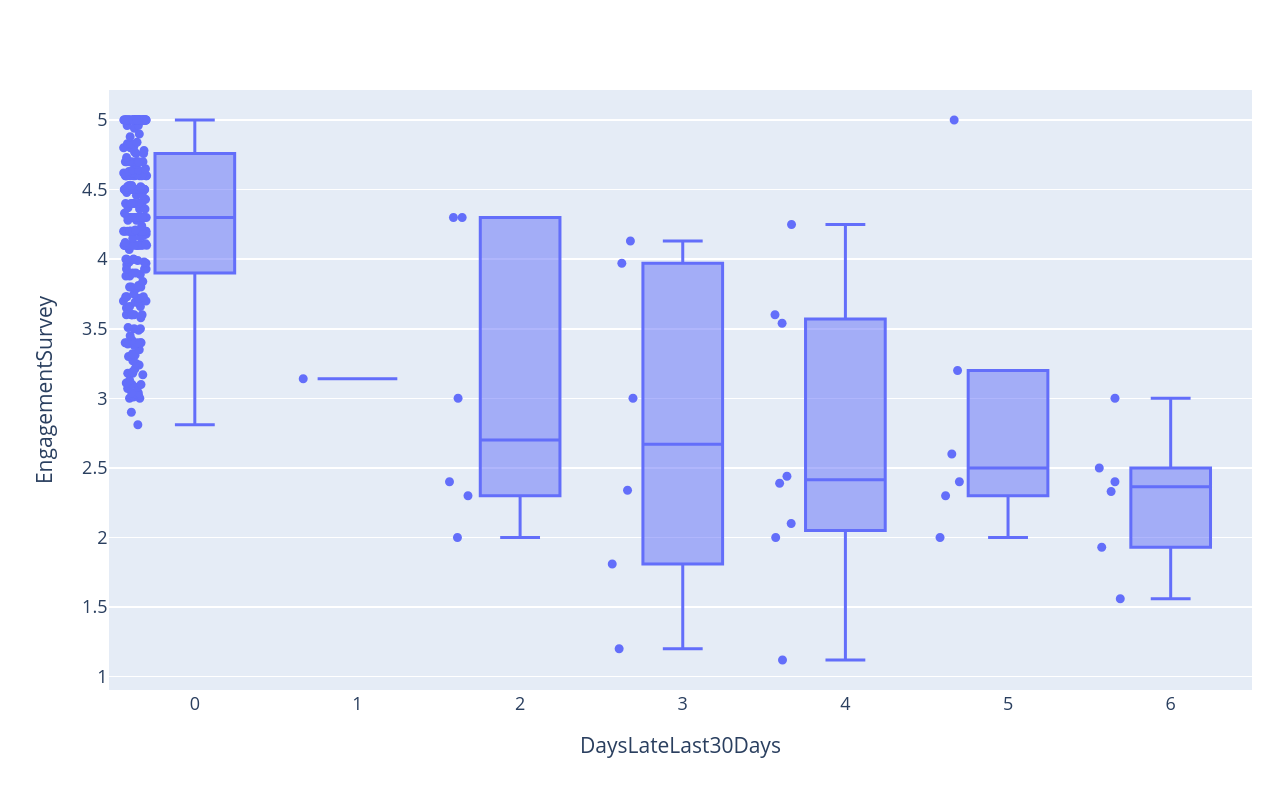
**Production and Sales and Software Engineering has the biggest proportion of poorer performers compared to others**

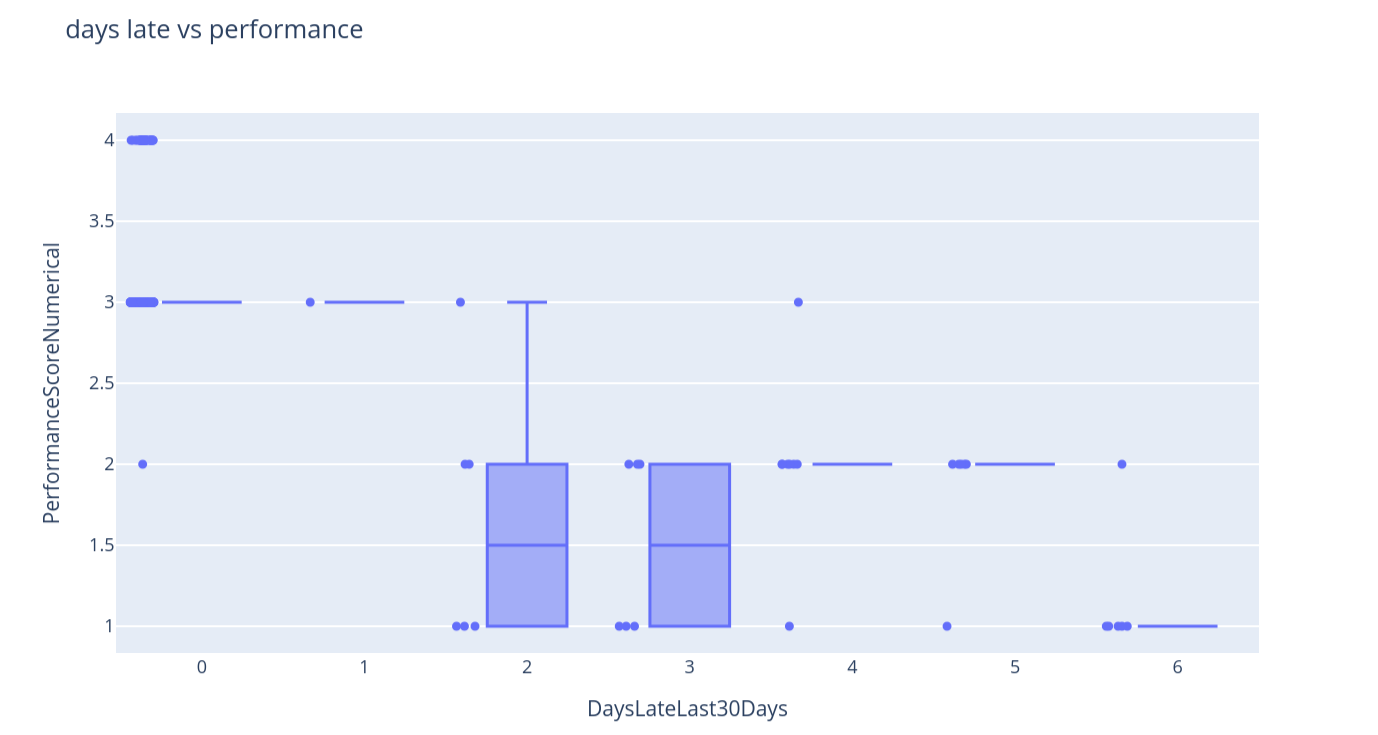


**Satsifaction and engagment are positively correlated to each other .**

**more employee satifaction and engagment seems to produces better preformances**

**Salary doesn’t seem to correlate to employee satifaction and engagment as much, there are many with low salaries that has high satisfaction and engagement**

**those that are not late at all for the past 30 days have at least higher engagement.**



## **more days late, the less performance**

## **Findings**

Some reforms should be done for production department, they have a lot of disengaged, unsatisfied and likely unhappy people in there that causes resignations and poorer performances. Is it because of the workload, managers or difficultly of the special projects involved?

Is the firing reasons of performance and absense with no good reasons based on their disengagement, unsatisfaction and unhappiness? More eda is needed on this

Sales and software engineering has similar issues too but with less resignations due to unhappiness unless

Boosting salary may not help in the satisfaction and engement metrics.

Days lates have that corelations to satifaction and engagment metric. do they really hate their jobs that they rather be late to work?

# **Modeling and evaluation for engagement and satisfaction**

creation of label that averages out for engagment and satisfaction As seen before some of the features are already encoded into "IDs"

\_feature\_names = [

# "EmployeeName",

# "EmployeeID",

"MarriedID",

"MaritalStatusID",

"GenderID",

"EmpStatusID",

"DepartmentID",

"PerformanceScoreNumerical",

"FromDiversityJobFairID",

"Salary",

"PositionID",

"Zip",

"DateOfBirth",

"CitizenDesc",

"DateOfHire",

"DateOfTermination",

# "TerminationReason",

"EmploymentStatus",

"ManagerID",

"EngagementSurvey",

"EmployeeSatisfaction",

"SpecialProjectsCount",

"LastPerformanceReviewDate",

"DaysLateLast30Days",

"Absences",

] # 22 features

print(len(\_feature\_names))

features = df.select(\_feature\_names)

labels = features["EmployeeSatisfaction"]

features = features.select(pl.exclude("EmployeeSatisfaction"))

sm = SMOTENC(

categorical\_features=features.select(pl.col(pl.String)).columns,

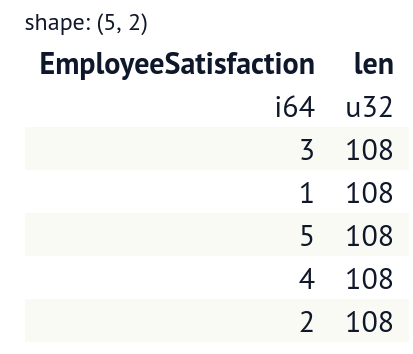
k\_neighbors=1,

)

(features, labels) = sm.fit\_resample(features.to\_pandas(), labels.to\_pandas())

features = pl.from\_pandas(features)

### **balanced out with SMOTE: a synthetic oversampling method**



**Random Forest Classifier with no hyperparameter tuning:**

metrics = pl.DataFrame()

ranks = pl.DataFrame()

pipeline = Pipeline(

[

(

"ordinal\_encoder\_column\_transformer",

ColumnTransformer(

[

(

"ordinal\_encoder",

OrdinalEncoder(),

features.select(pl.col(pl.String)).columns,

),

(

"passthrough",

"passthrough",

features.select(pl.exclude(pl.String)).columns,

),

]

),

),

("select\_k\_best", SelectKBest(k=11)),

("random\_forest\_classifier", RandomForestClassifier(random\_state=0)),

]

)

for i, (train\_index, test\_index) in enumerate(skfold.split(features, labels)):

test\_labels = labels[test\_index]

pipeline.fit(features[train\_index].to\_pandas(), labels[train\_index])

# print(features[test\_index].to\_pandas())

prediction = pipeline.predict(features[test\_index].to\_pandas())

prediction\_probabilities = pipeline.predict\_proba(

features[test\_index].to\_pandas()

)

t = pipeline.named\_steps["select\_k\_best"].get\_support()

names = features[t].columns

importance = pipeline.named\_steps[

"random\_forest\_classifier"

].feature\_importances\_

\_d = pl.DataFrame(

[names, importance], schema=["features", "importance"]

).sort("importance", descending=True)

ranks = ranks.hstack(

\_d["features"]

.to\_frame(name=str(i))

.with\_columns(rank=range(\_d.shape[0]))

.rename(dict(rank="rank" + str(i)))

.sort(str(i))

)

metrics = metrics.vstack(

pl.DataFrame(

{

"accuracy": accuracy\_score(test\_labels, prediction),

"recall": recall\_score(

test\_labels, prediction, average="macro"

),

"precision": precision\_score(

test\_labels, prediction, average="macro"

),

"auc": roc\_auc\_score(

test\_labels,

prediction\_probabilities,

average="macro",

multi\_class="ovr",

),

"pr\_auc": average\_precision\_score(

test\_labels, prediction\_probabilities, average="macro"

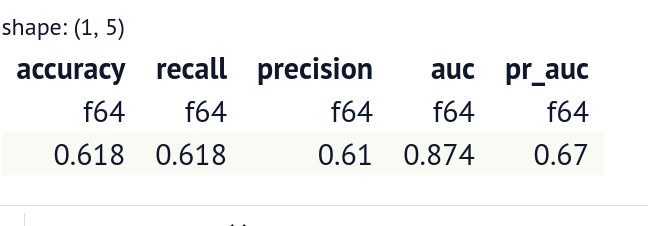
),

}

).with\_columns(pl.all().round(2))

)

**Metrics**



## **rank of feature importance on average**

1. "GenderID"
2. "PerformanceScoreNumerical"
3. "DepartmentID"
4. "EmploymentStatus"
5. "DateOfBirth"
6. "Absences"
7. "Salary"
8. "ManagerID"
9. "EngagementSurvey"
10. "DaysLateLast30Days"
11. "CitizenDesc"

## GridSearchCrossValidation attempt

With 3 fold cross validation. Grid search has cross validation in grained.

features\_train, features\_test, labels\_train, labels\_test = train\_test\_split(

features, labels, random\_state=0, test\_size=0.1

)

gs = GridSearchCV(

pipeline,

param\_grid={

"select\_k\_best\_\_k": [11, 16],

"random\_forest\_classifier\_\_n\_estimators": [50, 100],

"random\_forest\_classifier\_\_max\_depth": [3, 6, 9],

"random\_forest\_classifier\_\_max\_leaf\_nodes": [3, 6, 9],

},

)

gs.fit(features\_train, labels\_train)

print(gs.best\_estimator\_)

gs.predict\_proba(features[test\_index].to\_pandas())

**Output:**

Pipeline(steps=[('ordinal\_encoder\_column\_transformer', ColumnTransformer(transformers=[('ordinal\_encoder', OrdinalEncoder(), ['CitizenDesc', 'EmploymentStatus']), ('passthrough', 'passthrough', ['MarriedID', 'MaritalStatusID', 'GenderID', 'EmpStatusID', 'DepartmentID', 'PerformanceScoreNumerical', 'FromDiversityJobFairID', 'Salary', 'PositionID', 'Zip', 'DateOfBirth', 'DateOfHire', 'DateOfTermination', 'ManagerID', 'EngagementSurvey', 'SpecialProjectsCount', 'LastPerformanceReviewDate', 'DaysLateLast30Days', 'Absences'])])), ('select\_k\_best', SelectKBest(k=16)), ('random\_forest\_classifier', RandomForestClassifier(max\_depth=6, max\_leaf\_nodes=9, n\_estimators=50, random\_state=0))])

**Using the best parameter found using grid search**

\_metrics = pl.DataFrame()

\_ranks = pl.DataFrame()

\_pipeline = pipeline.set\_params(

\*\*{

"select\_k\_best\_\_k": 11,

"random\_forest\_classifier\_\_n\_estimators": 50,

"random\_forest\_classifier\_\_max\_depth": 6,

"random\_forest\_classifier\_\_max\_leaf\_nodes": 6,

}

)

\_pipeline.fit(features\_train, labels\_train)

\_prediction = \_pipeline.predict(features\_test)

\_prediction\_probabilities = \_pipeline.predict\_proba(features\_test)

\_t = \_pipeline.named\_steps["select\_k\_best"].get\_support()

\_names = features[t].columns

\_importances = \_pipeline.named\_steps[

"random\_forest\_classifier"

].feature\_importances\_

\_d = pl.DataFrame(

[\_names, \_importances], schema=["features", "importance"]

).sort("importance", descending=True)

ranks\_after\_gs = (

\_d["features"]

.to\_frame()

.with\_columns(rank=range(\_d.shape[0]))

.sort("rank", descending=True)

)

metrics\_after\_gs = pl.DataFrame(

{

"accuracy": accuracy\_score(labels\_test, \_prediction),

"recall": recall\_score(labels\_test, \_prediction, average="macro"),

"precision": precision\_score(

labels\_test, \_prediction, average="macro"

),

"auc": roc\_auc\_score(

labels\_test,

\_prediction\_probabilities,

average="macro",

multi\_class="ovr",

),

"pr\_auc": average\_precision\_score(

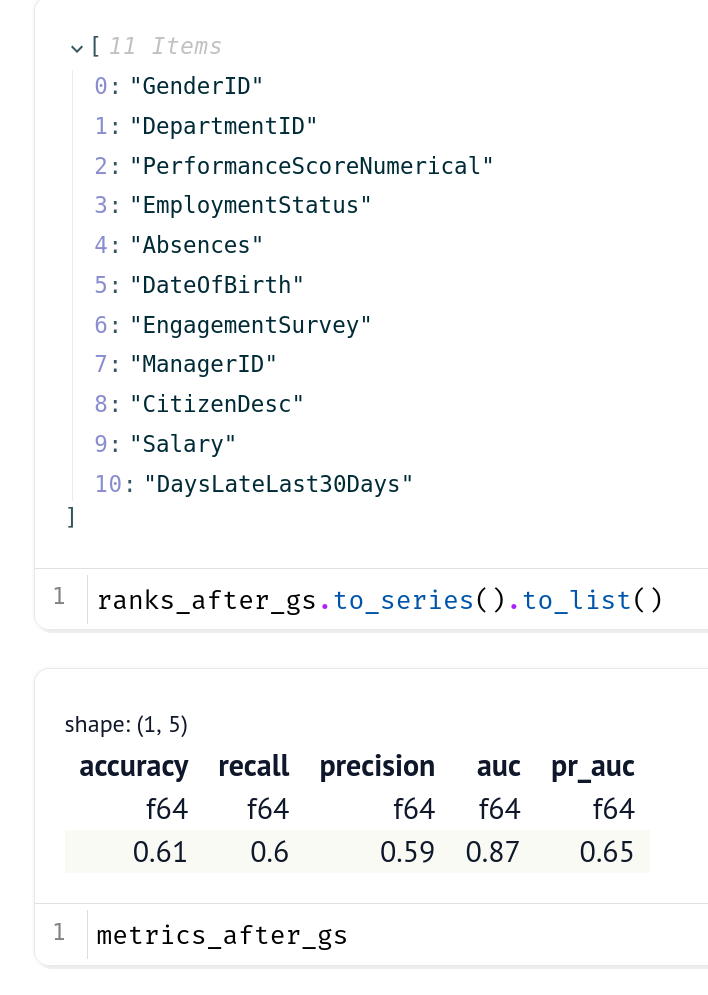
labels\_test, \_prediction\_probabilities, average="macro"

),

}

).with\_columns(pl.all().round(2))

**Ranks and metrics after gridsearch**



There are a bit of improvement overall after grid search

**Other plans**

Maybe more feature engineering would help in the improvement in the model peformance. Also I did not do any hypothesis testing of the features to target. I will probably use Test of normality for numerical features (like Sapiro for example) , test of of variance (like levene for example)and test of mean or variance (T, F, U or etc tests). The sub category of test fro each of these tests will depend on the nature of the distribution like it’s parameters.

# With the introduction of Artificial Intelligence and Large Language Model how do you think the roles of data analyst and data scientist will be affected?

More automation so that data analysts and scientist could focus work on the insights and predictions . But LLMs prone to hallucinations and not so correct findings so far even with RAG and finetuning. And understanding stakeholders are important too which I don’t think these AI are not there yet in handling them.