The Lightweight IBM Cloud Garage Method for Data Science

Architectural Decisions Document

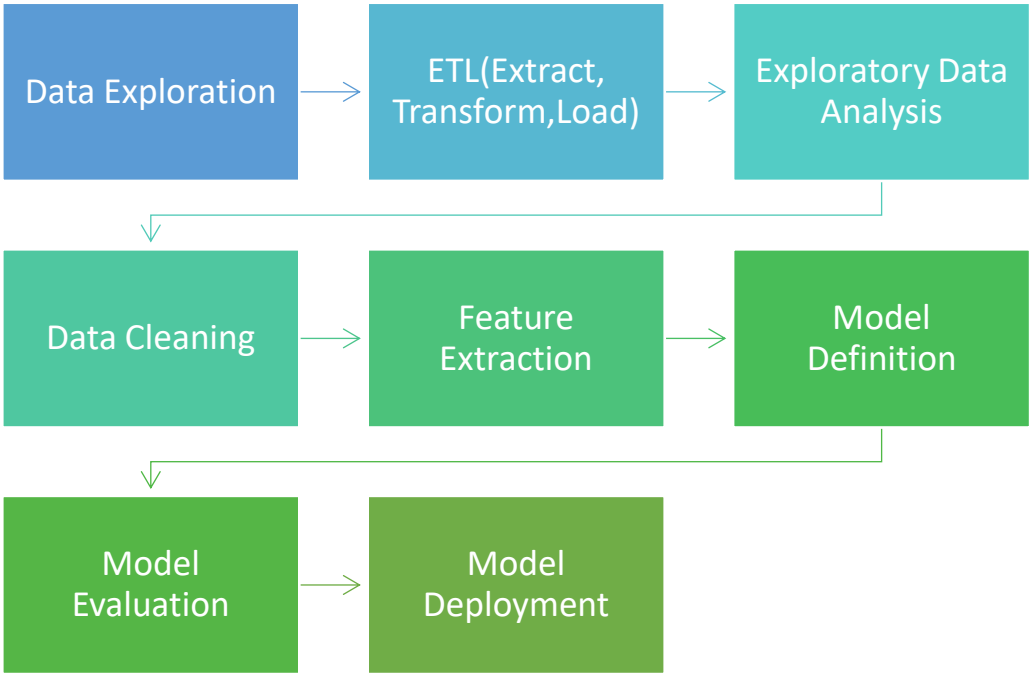
Project Name: Employee Attrition Analysis

Data Source: Kaggle

Link to the dataset: [Employee Satisfaction Survey Data (kaggle.com)]

Every task has a clear purpose and a defined work product (for example, a Jupyter Notebook, a script, or a docker container hosting a scoring or training endpoint, depending on the architectural decisions made).

The steps involved in the analysis of this project are:



This Architectural Decision Document explains every process taken in the project. In case of any doubts about the project go through this document.

1. Data Exploration

   The Employee Satisfaction Survey dataset is a comprehensive collection of information regarding employees within a company. It includes essential details such as employee identification numbers, self-reported satisfaction levels, performance evaluations, project involvement, work hours, tenure with the company, work accidents, promotions received in the last 5 years, departmental affiliations, and salary levels. This dataset offers valuable

insights into the factors influencing employee satisfaction and can be used to analyze and understand various aspects of the workplace environment.

2. ETL – Extract Transform Load

```
#pandas.read_csv => used to read a csv data file
df = pd.read_csv('/kaggle/input/employees-satisfaction-analysis/Employee Attriti
#show the first 10 rows of the imported data
df.head(10)
```

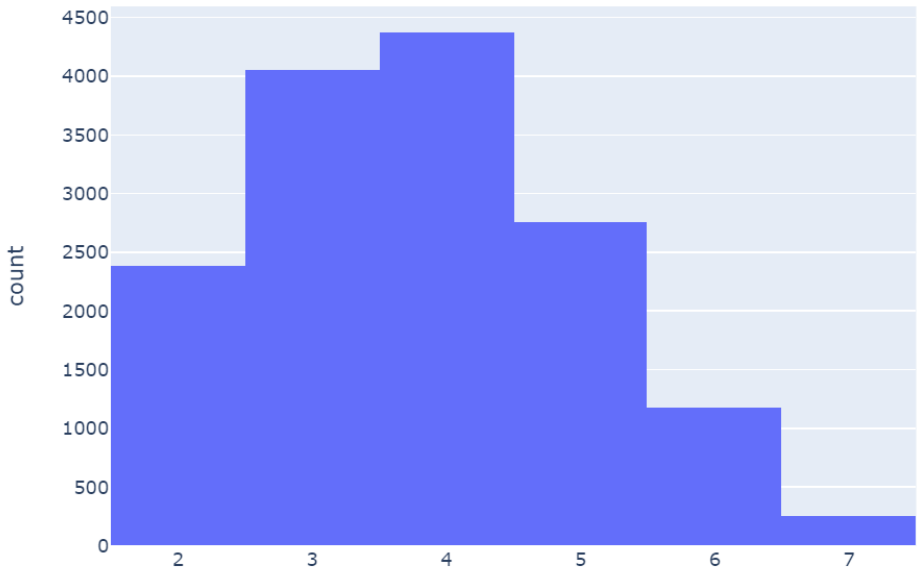
The first and most important process of data analysis is to explore the data and find out the required information for the analysis.
Extract, transform, and load (ETL) is the process of combining data from multiple sources into a large, central repository called a data warehouse. ETL uses a set of business rules to clean and organize raw data and prepare it for storage, data analytics, and machine learning (ML).

3. EDA – Exploratory Data Analysis

Learn everything you need to know about exploratory data analysis, a method used to analyze and summarize data sets. Exploratory data analysis (EDA) is used by data scientists to analyze and investigate data sets and summarize their main characteristics, often employing data visualization methods.

The methods used in this project for EDA are

Histogram – for analysing the distribution of project among employees



Box lot – to detect the outliers present in the data

Various statistical inferences for finding the relationship between satisfaction level and other factors.

4. Data Cleaning

Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. When combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled.

Dealing with missing values and handling it

```
#finding the missing values
missing_values = df[df.isna().any(axis=1)]
display(missing_values.head())
display(missing_values.isna().count())
```

```
#there are 788 entries with missing values. Since all those 788 rows does not ha
df.dropna(inplace=True)
```

5. Feature Extraction

In order to find the best features, we have to understand the dependency of each features on the dependent variable or target vector. Correlation matrix is the most common way of finding the best feature.

The below images are the screenshots of my project. The steps give better understanding on how feature extraction is done.

```
plt.figure(figsize = (30,15))
sns.heatmap(df2.corr(method="pearson"),annot = True)
sns.set(font_scale = 1.5)
```

Label encoding for categorical data for Model Development

```
#label encoding for categorical data
labelEncoder = LabelEncoder()
for value in categoricalColumns:
    df[value] = labelEncoder.fit_transform(df[value])
```

General Regression analysis equation is given as

$$Y = \beta_0 + \beta X$$

$y => TargetVector$

$x => Feature$

Here since the X is array of features we use

$$Y = \beta_i^T . X_i + \beta_0$$

$X_i => featureVectors\ \beta_0 => Bias\ \beta T_i => Weights\ Y => TargetVector$

```
#setting up of target variable and feature variable
newDf = df.copy()

#let target = y,  and features = X
y = newDf['satisfaction_level']
display(y)

newDf.drop('satisfaction_level',axis=1,inplace=True)
X = newDf
display(X)
```

6. Model Development or model definition

Finding the appropriate model is the next step in data analysis. Here we use DecisionTreeRegressor as the model for regression.

```
#splitting the dataset to train_test_split
X_train,X_test, y_train,y_test = train_test_split(X,y,shuffle = True)
```

```
tree = DecisionTreeRegressor(max_depth=8,random_state=40)
tree.fit(X_train,y_train)
```

```
DecisionTreeRegressor(max_depth=8, random_state=40)
```

7. Model Evaluation

```
print(tree.score(X_train,y_train))
print(tree.score(X_test,y_test))
```

```
0.4866259323250689
0.41808934753158133
```

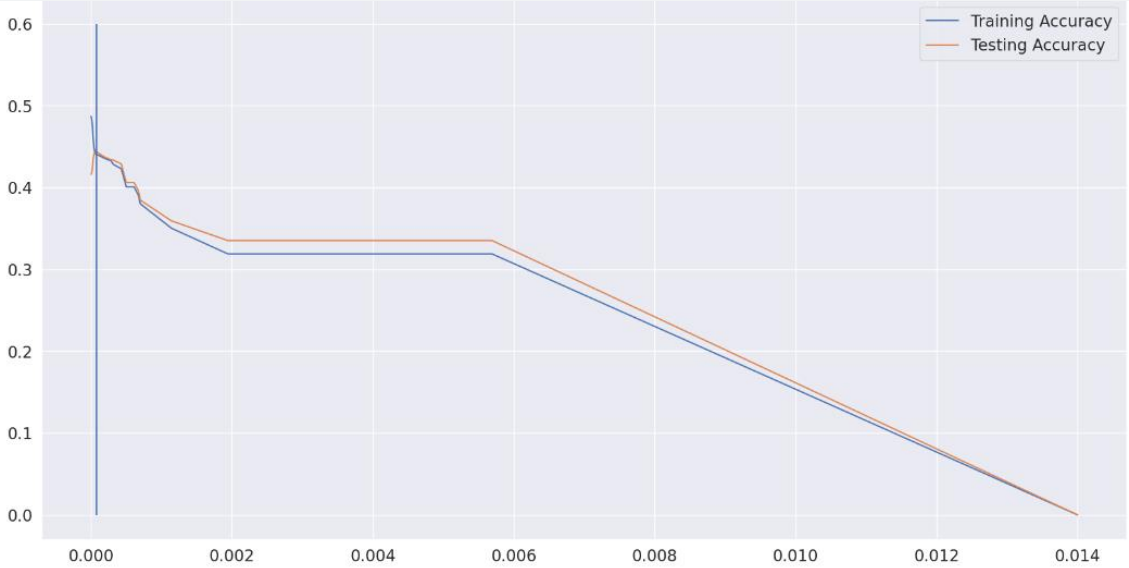
Here we can see there is a variation in training set score and testing set score. Basically it called as overfitting. So we have to deal with overfitting. One of the most common way of dealing with over fitting in decision tree regression is by a method called pruning.

```
# applying cost complexity pruning inorder to reduce the overfitting
path = tree.cost_complexity_pruning_path(X_train,y_train)
alphas = path['ccp_alphas'].round(5)
print(alphas)
```

```
train_score,test_score = [],[]
for alpha in alphas:
    decisionTree = DecisionTreeRegressor(ccp_alpha = alpha,max_depth=8)
    decisionTree.fit(X_train,y_train)

    train_score.append(decisionTree.score(X_train,y_train))
    test_score.append(decisionTree.score(X_test,y_test))
```

```
fig = plt.figure(figsize=(20,10))
sns.lineplot(x=alphas, y=train_score,label="Training Accuracy")
sns.lineplot(x=alphas, y=test_score,label="Testing Accuracy")
plt.vlines(alphas[np.argmax(test_score)],0,0.6)
fig.show()
```



From the plot the training score and testing score is approximately same when alpha is between 0.001 and 0.002. Let's take that alpha and then evaluate the model

```
#updated test_score and training score

print(dtree.score(X_train,y_train))
print(dtree.score(X_test,y_test))
```

```
0.44012934305338036
0.44297443770299105
```

Here we could see it is approximately same. So overfitting issue has been resolved

```python
#R-squared Value
print(f"The R-squared value : {dtree.score(X_test,y_test):.2%}")
```

```
The R-squared value : 44.30%
```

+ Code + Markdown