DATA PREPARATION

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
```

```
main_data = pd.read_csv('Attrition_data.csv')
```

```
main_data.head()
```

```
location_clean = pd.read_csv('location_clean.csv')
location_clean.head()
```

+ Code + Text

```
data = pd.merge(main_data, location_clean, how= 'inner',left_on = 'S.No', right_on='id' )
data.drop('id',axis =1, inplace = True)
data.head()
```

```
assert location_clean.shape[0] == data.shape[0]
```

```
data.shape
```

Converting the columns into right datatypes and extracting data

```
data = data.rename(columns = {'Engagement Score (% Satisfaction)':'sat_score'})
data['sat_score'] = data['sat_score'].apply(lambda x:x[:-1])
data.head()
```

```
data['sat_score'] = data['sat_score'].astype('int')
```

```
data.isna().sum().sum()
```

```
data.isna().sum()
```

```
data = data[data['doubtful']=='NO']
data.isna().sum()
```

```
np.where(data.isna())
```

```
data.iloc[[  2,  23,  63, 193],:]
```

```
data.info()
```

```
data['Last Rating'] = data['Last Rating'].apply(lambda x: str(x))
data.info()
```

```
def to_float(x):
    try:
        return float(x)
    except ValueError as v:
        return float(x.replace(' ',''))
data['Tenure'] = data['Tenure'].apply(to_float)
data.head(10)
```

```
from datetime import datetime
def converter(x):
    try:
        return datetime.strptime(x, '%d-%b-%y')
    except:
        return datetime.strptime(x, '%d-%m-%y')  # for these values in DOJ column '''05-07-10

data['DOL_date'] = data['In Active Date'].apply(converter)
data['DOJ_date'] = data['DOJ'].apply(converter)
data.head()
```

```
data.drop(['DOJ','In Active Date'], axis = 1, inplace =True)
```

```
data['Designation'].value_counts()
```

```
data['Grade'].value_counts()
```

```
data.groupby('Designation')['Grade'].apply(lambda x: x.unique())
```

```
data['Zone'].value_counts()
```

```
data['Zone'] = data['Zone'].apply(lambda x: x.lower()) ## CENTRAL and central, north and Nort
data['Zone'].value_counts()
```

```
data['Marital Status'].value_counts()
```

```
data['Gender'].value_counts()
```

```
data['Education'].value_counts()
```

```
data.columns
```

```
data.drop(['EmpID','Emp Name','Attrition ','Designation'],axis =1 , inplace =True)
data.head()
```

Feature Engineering

```
data['tenure_days'] = (data['DOL_date'] - data['DOJ_date']).apply(lambda x:x.days)
data.head()
```

```
data.columns
```

```
data = data.rename(columns = {'S.No':'id', 'Last Rating':'rating','Monthly Income':'income','
data.head()
```

```
data.columns = [col.lower() for col in data.columns]
data.head()
```

```
data['location'].isna().sum()
```

```
data.columns
```

```
data = data.drop(['doubtful','location','changed'] ,axis = 1)
```

```
data.head()
data.shape
```

```
data.to_csv('data_complete_location.csv', index= False)
```

EXPLORATORY DATA ANALYSIS

Univariate visualization

```
numeric_col = [col for col in data.columns if data[col].dtype in ['int64','int32','float64']
```

```
numeric_col
```

```
data[numeric_col].hist(figsize=(16, 8));
```

```
_, axes = plt.subplots(nrows=2, ncols=2, figsize=(16, 8))
i = 0
j = 0
for col in numeric_col:
    _=sns.distplot(data[col], ax=axes[i][j]);
    _=plt.xticks(rotation=90)
    j+=1
    if j==2:
        i+=1
        j=0
```

```
_, axes = plt.subplots(nrows=2, ncols=2, figsize=(16, 8))
i = 0
j = 0
for col in numeric_col:
    _=sns.boxplot(data[col], ax=axes[i][j]);
    _=plt.xticks(rotation=90)
    j+=1
    if j==2:
        i+=1
        j=0
```

```
cat_cols = [col for col in data.columns if data[col].dtype == 'object']
cat_cols
```

```
%matplotlib inline
_, axes = plt.subplots(nrows=5, ncols=2,sharey=True, figsize=(16, 24))
# plt.subplot_tool() ## Works for interactive
plt.subplots_adjust(hspace=0.8)
i = 0
j = 0
for col in cat_cols:
    if col == 'location': continue
    g=sns.countplot(x=col, data=data, ax=axes[i][j], order = list(data[col].value_counts().re
    if col in  ['remarks','corrected_location','district','state']:
        _=g.set_xticklabels(g.get_xticklabels(), rotation=90)
#     _ = plt.xticks(rotation=90)
    j+=1
    if j==2:
        i+=1
        j=0
```

Multivariate visualization

```
corr_matrix = data[numeric_col].corr()
sns.heatmap(corr_matrix, annot = True);
```

```
numeric_col
```

```
cat_cols
```

```
_, axes = plt.subplots(nrows=5, ncols=2,sharey=True, figsize=(16, 30))
plt.subplots_adjust(hspace=0.8)

i = 0
j = 0
for col in cat_cols:
    if col == 'location': continue
    g=sns.boxplot(x=col,y='tenure_days', data=data, ax=axes[i][j]);
    if col in  ['remarks','corrected_location','district','state']:
        _=g.set_xticklabels(g.get_xticklabels(), rotation=90)
#     _ = plt.xticks(rotation=90)
    j+=1
    if j==2:
        i+=1
        j=0
```

```
plt.figure(figsize= (8,6))
sns.boxplot(x='grade', y='income', data=data[data['income']<1e5], order = sorted(data['grade'
```

CLUSTERING

```
%reset -f
```

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
```

```
data = pd.read_csv('data_complete_location.csv')
data.head()
```

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
# Filter data
```

```
left_emp =  data[['sat_score', 'rating']]
# Create groups using K-means clustering.

ss= StandardScaler()
left_emp_scaled = ss.fit_transform(left_emp)
left_emp_scaled.shape
kmeans = KMeans(n_clusters = 4, random_state = 10).fit(left_emp_scaled)
```

```
left_emp['label'] = kmeans.labels_
# Draw scatter plot
_ = plt.scatter(left_emp['sat_score'], left_emp['rating'], c=left_emp['label'],cmap='Accent')
_ = plt.xlabel('Satisfaction Level')
_ = plt.ylabel('Last Evaluation')
_ = plt.title('4 Clusters of employees who left')
plt.show()
```

```
left_emp =  data[['tenure', 'income']]
left_emp = left_emp[left_emp['income']<1e5]
# Create groups using K-means clustering.

ss= StandardScaler()
left_emp_scaled = ss.fit_transform(left_emp)
left_emp_scaled.shape
kmeans = KMeans(n_clusters =4 , random_state = 10).fit(left_emp_scaled)

left_emp['label'] = kmeans.labels_
# Draw scatter plot
_ = plt.scatter(left_emp['tenure'], left_emp['income'], c=left_emp['label'],cmap='Accent')
_ = plt.xlabel('Tenure')
_ = plt.ylabel('Income')
_ = plt.title('4 Clusters of employees who left')
plt.show()
```

```
left_emp =  data[['age', 'income']]
left_emp = left_emp[left_emp['income']<1e5]
# Create groups using K-means clustering.

ss= StandardScaler()
left_emp_scaled = ss.fit_transform(left_emp)
left_emp_scaled.shape
kmeans = KMeans(n_clusters =6 , random_state = 10).fit(left_emp_scaled)

left_emp['label'] = kmeans.labels_
# Draw scatter plot
_=plt.scatter(left_emp['age'], left_emp['income'], c=left_emp['label'],cmap='Accent')
_=plt.xlabel('Age')
_=plt.ylabel('Income')
_=plt.title('6 Clusters of employees who left')
plt.show()
```

FREQUENT PATTERN MINING

```
%reset -f
```

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
```

```
data = pd.read_csv('data_complete_location.csv')
data.head()
```

```
grade_int = {'E1':1,'E2':2,'M1':3,'M2':4,'M3':5,'M4':6,'CXO':7}
data['grade_int'] = data['grade'].apply(lambda x: grade_int[x])
```

```
not_required =  ['grade','dol_date','doj_date','id','corrected_location','district']
selected_cats = [ col for col in data.columns if data[col].dtype=='object' and col not in not
selected_cats
```

```

```

```
selected_nums = [col for col in data.columns if col not in selected_cats+not_required]
selected_nums
```

Frequent Item Sets Some points to be noted:

Income is dependent on the grade of the employee. Age and income are positively correlated Due to the above two points, only grade is considered for the frequent item sets calculation Tenure and sat_score are binned so as to be used for frequent itemset calculation purpose.

```
sns.distplot(data['tenure'])
sns.distplot(data['sat_score'])
```

```
def sat_binner(x):
    return x//20 + 1 if not x%20 == 0 else x//20
data['sat_binned'] = data['sat_score'].apply(sat_binner).astype('object')
```

```
def tenure_binner(x):
```

```
    return x//2 + 1 if not x%2 == 0 else x//2
data['tenure_binned'] = data['tenure'].apply(tenure_binner).astype('object')
```

```
cols_for_frequent_items = ['grade','gender','education','rating','marital_status','zone','rem
data_fp = data[cols_for_frequent_items]
data_fp_enc = pd.get_dummies(data_fp, columns = data_fp.columns)
data_fp_enc.head()
```

```
pd.set_option('max_colwidth', 100)
```

```
pip install mlxtend
```

```
from mlxtend.frequent_patterns import apriori

freq_pattern = apriori(data_fp_enc, min_support=0.20, use_colnames=True)
freq_pattern['length'] = freq_pattern['itemsets'].apply(lambda x: len(x) )
freq_pattern[freq_pattern['length']>=4].sort_values('support',ascending= False)
```

```
fp2 = data[(data['gender']== 'Male') & (data['grade']=='E1') & (data['education'] =='Bachelor
fp2.groupby('remarks').size().sort_values(ascending = False)
```

```
# Interesting FP
fp1 = data[(data['gender']== 'Male') & (data['grade']=='E1') & (data['education'] =='Bachelor
```

TENURE PREDICTION

```
%reset -f
```

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

```
from sklearn.preprocessing import StandardScaler
from xgboost import XGBRegressor
from sklearn.linear_model import LinearRegression, Lasso, Ridge
# from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.dummy import DummyRegressor
# from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

```
from math import sqrt
np.random.seed(42)
```

```
data = pd.read_csv('data_complete_location.csv')
data.head()
```

```
data_pred = data[['id','grade','tenure','gender','education','age','rating','income','sat_sco
                  'zone','remarks']]
```

```
X = data_pred.drop(['id','tenure'], axis =1)
y = data_pred['tenure']
```

```
selected_cats = [col for col in X.columns if X[col].dtype == 'object']
selected_nums = [col for col in X.columns if col  not in selected_cats]
```

```
X = pd.get_dummies(X, columns = selected_cats)
train_samples = int(0.9*data_pred.shape[0])
train_indices = list(range(train_samples))
val_indices = list(range(train_samples, data_pred.shape[0]))
train_X = X.loc[train_indices, : ]
train_y = y.loc[train_indices]
val_X = X.loc[val_indices, : ]
val_y = y.loc[val_indices]

train_X.shape
train_y.shape
val_X.shape
val_y.shape
```

```
ss= StandardScaler()
train_X_scaled = pd.DataFrame(ss.fit_transform(train_X), columns = train_X.columns)
train_y_logged = np.log1p(train_y)
val_X_scaled = pd.DataFrame(ss.transform(val_X), columns = val_X.columns)
# val_y = np.log1p(val_y)
```

```
def fit_model(model):
    if model == DecisionTreeRegressor:
        reg = model(random_state = 291)
    else:
        reg = model()
    reg.fit(train_X_scaled, train_y_logged)
    val_y_hat = np.expm1(reg.predict(val_X_scaled))
    print(f'MAE: {mean_absolute_error(val_y_hat, val_y)}')
    print(f'RMSE: {sqrt(mean_squared_error(val_y_hat, val_y))}')
#     return sqrt(mean_squared_error(val_y_hat, val_y))
```

```
    fig, ax = plt.subplots(1,2, figsize=(16,4))

    ax[0].plot(list(range(len(val_y))), val_y_hat, label= 'Predicted Tenure (in yrs)')
    ax[0].plot(list(range(len(val_y))), val_y, label = 'Original  Tenure (in yrs)')
    ax[0].legend(loc = 'best')
    ax[0].set_title('Predictions')

    print(f'Using model : {model}')
    if model in [Lasso, Ridge, LinearRegression]:
        coeff_df = pd.DataFrame(reg.coef_, train_X_scaled.columns, columns=['Coefficient'])

    elif model in [XGBRegressor,DecisionTreeRegressor]:
        coeff_df = pd.DataFrame(reg.feature_importances_, train_X_scaled.columns, columns=['C

    else:
        print("No feature importance graph for DummyRegressor")
        return

    coeff_df["abs"] = coeff_df.Coefficient.apply(np.abs)
    coeff_df = coeff_df.sort_values(by="abs", ascending=False).drop("abs", axis=1)

    ax[1].bar(coeff_df.index[:15],coeff_df['Coefficient'][:15])
    _ = plt.xticks(rotation=90)
    ax[1].set_title('Feature importance')
```

```
fit_model(DummyRegressor)
```

```
fit_model(LinearRegression)
```

```
fit_model(XGBRegressor)
```

```
def plot_ensemble(model1, model2):
    if model1 == DecisionTreeRegressor:
        m1  = model1(random_state=291)
    else:
        m1 = model1()
    m1.fit(train_X_scaled, train_y_logged)
    m2 = model2()
    m2.fit(train_X_scaled, train_y_logged)
    val_y_hat = (np.expm1(m1.predict(val_X_scaled)) + np.expm1(m2.predict(val_X_scaled)))/2.0
    print(f'MAE: {mean_absolute_error(val_y_hat, val_y)}')
    print(f'RMSE: {sqrt(mean_squared_error(val_y_hat, val_y))}')

    fig, ax = plt.subplots(1,1)

    ax.plot(list(range(len(val_y))), val_y_hat, label= 'Predicted Tenure (in yrs)')
```

```
    ax.plot(list(range(len(val_y))), val_y, label = 'Original  Tenure (in yrs)')
    ax.legend(loc = 'best')
    ax.set_title('Predictions')
```

```
plot_ensemble(LinearRegression, XGBRegressor)
```

```
plot_ensemble(DecisionTreeRegressor, XGBRegressor)
```