PROJECT REPORT ON

"HUMAN RESOURCE MANAGEMENT : PREDICTING EMPLOYEE PROMOTIONS USING MACHINE LEARNING"

1) Project Planning and Initialization Phase –

PROBLEM STATEMENT :-

The aim is to analyse the various factors that can contribute to the promotion of an employee. Based on the analysis, predict which employees will be promoted.

PROPOSED SOLUTION :-

Analyse the various factors that can contribute to the promotion of an employee. Based on the analysis, predict which employees will be promoted, using machine learning algorithms.

INITIAL PROJECT PLANNING REPORT :-

1st step – Data Collection
2nd step – Data Preparation
- Data Mining Classification Process using C.4.5
- Data Mining Classification Process using RF
- Data Mining Classification Process using SVM
- Data Mining Classification Process using Naïve Bayes

3rd step – Data Mining Evaluation
4th step – Highest accuracy specificity and Sensitivity total
5th step – Classification Roles

2) Data Collection and Preprocessing Phase –

```
[2]: import numpy as np
     import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt
```

```
[5]: data =pd.read_csv('emp_promotion.csv')
```

3) Visualizing and Analysing the Data –

```
data =pd.read_csv('emp_promotion.csv')
```

```
data.head()
```

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	award
0	65438	Sales & Marketing	region_7	Master's & above	f	sourcing	1	35	5.0	8	1	
1	65141	Operations	region_22	Bachelor's	m	other	1	30	5.0	4	0	
2	7513	Sales & Marketing	region_19	Bachelor's	m	sourcing	1	34	3.0	7	0	
3	2542	Sales & Marketing	region_23	Bachelor's	m	other	2	39	1.0	10	0	
4	48945	Technology	region_26	Bachelor's	m	other	1	45	3.0	2	0	

```
plt.figure(figsize =(10,4))
```

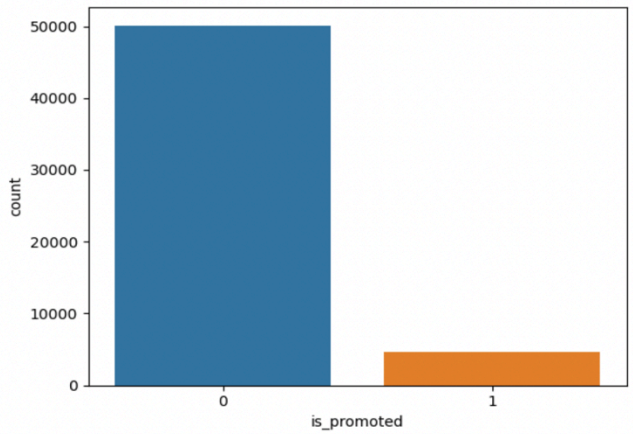
```
<Figure size 1000x400 with 0 Axes>
<Figure size 1000x400 with 0 Axes>
```

Univariate Analysis –

Univariate Analysis

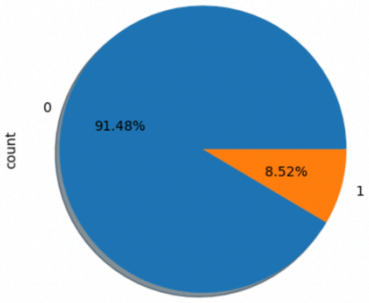
[8]:
```
sns.countplot(x='is_promoted',data=data)
```

[8]:
```
<Axes: xlabel='is_promoted', ylabel='count'>
```



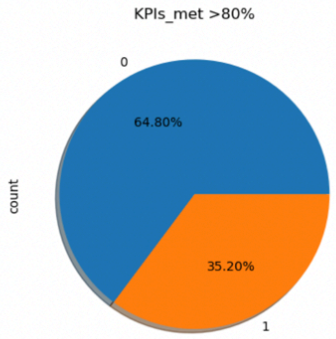
[9]:
```
data['is_promoted'].value_counts().plot(kind="pie", autopct ="%.2f%%",shadow=True)
plt.show
```

[9]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



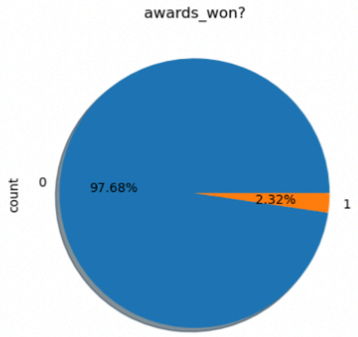
[11]:
```
data['KPIs_met >80%'].value_counts().plot(kind="pie",autopct ="%.2f%%",shadow=True)
plt.title('KPIs_met >80%')
```

[11]: Text(0.5, 1.0, 'KPIs_met >80%')



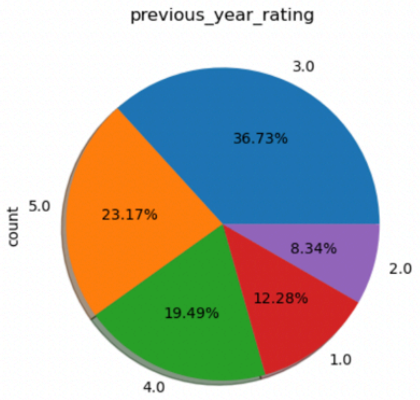
[12]:
```
data['awards_won?'].value_counts().plot(kind="pie",autopct ="%.2f%%",shadow=True)
plt.title('awards_won?')
```

[12]: Text(0.5, 1.0, 'awards_won?')



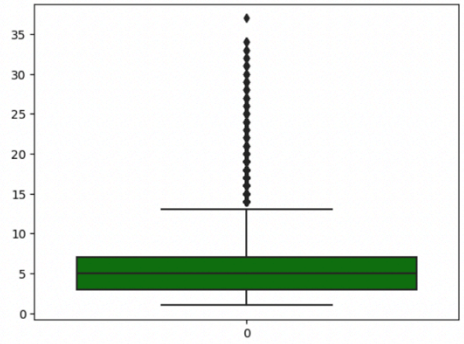
[13]:
```
data['previous_year_rating'].value_counts().plot(kind="pie",autopct ="%.2f%%",shadow=True)
plt.title('previous_year_rating')
```

[13]: Text(0.5, 1.0, 'previous_year_rating')



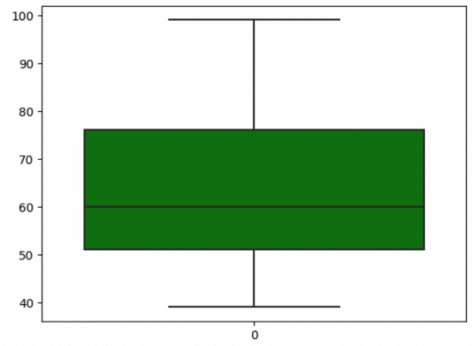
```
[15]: sns.boxplot(data['length_of_service'],color='g')
```

```
[15]: <Axes: >
```




```
[16]: sns.boxplot(data['avg_training_score'],color='g')
```

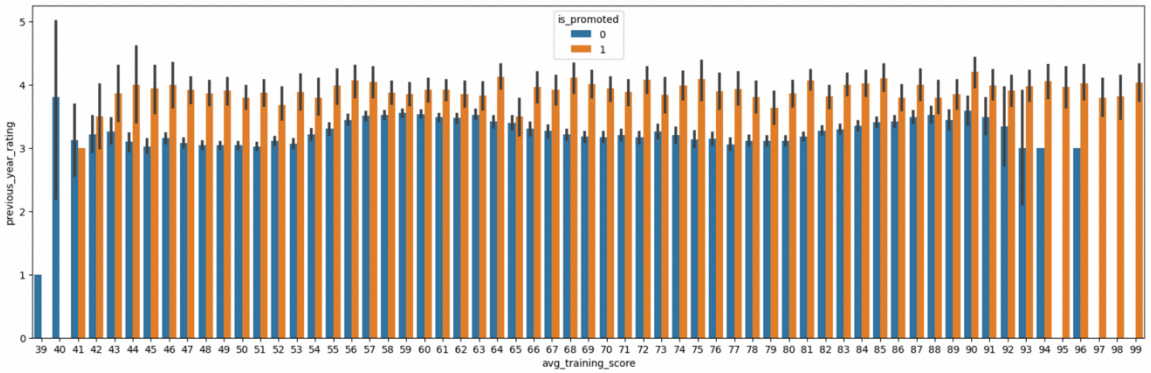
```
[16]: <Axes: >
```



Multivariate Analysis –

Multivariate analysis

[113]:
```
data['is_promoted'] = data['is_promoted'].astype(str)
plt.figure(figsize=(20, 6))
sns.barplot(x='avg_training_score', y='previous_year_rating', hue='is_promoted', data=data)
plt.show()
```



Descriptive Analysis

[19]:
```
data.describe(include='all')
```

[19]:

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	5
count	54808.000000	54808	54808	52399	54808	54808	54808.000000	54808.000000	50684.000000	54808.000000	
unique	NaN	9	34	3	2	3	NaN	NaN	NaN	NaN	
top	NaN	Sales & Marketing	region_2	Bachelor's	m	other	NaN	NaN	NaN	NaN	
freq	NaN	16840	12343	36669	38496	30446	NaN	NaN	NaN	NaN	
mean	39195.830627	NaN	NaN	NaN	NaN	NaN	1.253011	34.803915	3.329256	5.865512	
std	22586.581449	NaN	NaN	NaN	NaN	NaN	0.609264	7.660169	1.259993	4.265094	
min	1.000000	NaN	NaN	NaN	NaN	NaN	1.000000	20.000000	1.000000	1.000000	
25%	19669.750000	NaN	NaN	NaN	NaN	NaN	1.000000	29.000000	3.000000	3.000000	
50%	39225.500000	NaN	NaN	NaN	NaN	NaN	1.000000	33.000000	3.000000	5.000000	
75%	58730.500000	NaN	NaN	NaN	NaN	NaN	1.000000	39.000000	4.000000	7.000000	
max	78298.000000	NaN	NaN	NaN	NaN	NaN	10.000000	60.000000	5.000000	37.000000	

Drop unwanted features

[20]:
```
data.drop(['employee_id','gender','region','recruitment_channel'],axis=1)
```

[20]:

	department	education	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_score	is_promoted
0	Sales & Marketing	Master's & above	1	35	5.0	8	1	0	49	0
1	Operations	Bachelor's	1	30	5.0	4	0	0	60	0
2	Sales & Marketing	Bachelor's	1	34	3.0	7	0	0	50	0
3	Sales & Marketing	Bachelor's	2	39	1.0	10	0	0	50	0
4	Technology	Bachelor's	1	45	3.0	2	0	0	73	0
...	...	...	...	...	...	...	...	...	...	...
54803	Technology	Bachelor's	1	48	3.0	17	0	0	78	0
54804	Operations	Master's & above	1	37	2.0	6	0	0	56	0
54805	Analytics	Bachelor's	1	27	5.0	3	1	0	79	0
54806	Sales & Marketing	NaN	1	29	1.0	2	0	0	45	0
54807	HR	Bachelor's	1	27	1.0	5	0	0	49	0

54808 rows × 10 columns

checking for null values

[21]:
```
data.isnull().sum()
```

[21]:
```
employee_id               0
department                0
region                    0
education              2409
gender                    0
recruitment_channel       0
no_of_trainings           0
age                       0
previous_year_rating   4124
length_of_service         0
KPIs_met >80%             0
awards_won?               0
avg_training_score        0
is_promoted               0
dtype: int64
```

[22]:
```
#handling null values
data['education'].fillna(data['education'].mode()[0],inplace=True)
data['previous_year_rating'].fillna(data['previous_year_rating'].mode()[0],inplace=True)
```

[23]:
```
data.isnull().sum()
```

[23]:
```
employee_id           0
department            0
region                0
education             0
gender                0
recruitment_channel   0
no_of_trainings       0
age                   0
previous_year_rating  0
length_of_service     0
KPIs_met >80%         0
awards_won?           0
avg_training_score    0
is_promoted           0
dtype: int64
```

Remove negative values

[24]:
```
negative =data[(data['KPIs_met >80%']==0) & (data['awards_won?']==0) & (data['previous_year_rating']==1.0) & (data['is_promoted']==1) & (data[
```

[25]:
```
negative.head()
```

[25]:

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	a
31860	29663	Sales & Marketing	region_22	Bachelor's	m	referred	1	27	1.0	2	0	
51374	28327	Sales & Marketing	region_2	Bachelor's	m	sourcing	1	31	1.0	5	0	

[26]:
```
data.drop(index=[31860,51374],inplace=True)
```

handling outliers

[27]:
```
q1 = np.quantile(data['length_of_service'],0.25)
q3 = np.quantile(data['length_of_service'],0.75)
```

[28]:
```
IQR = q3-q1
IQR
```

[28]:
```
4.0
```

[29]:
```
upperBound =(IQR*1.5)+q3
lowerBound =(IQR*1.5)-q1
```

[30]:
```
print('q1 =',q1)
print('q3 =',q3)
print('IQR =',IQR)
print('upperBound =',upperBound)
print('lowerBound =',lowerBound)
print('skewed data =',len(data[data['length_of_service']> upperBound]))
```

```
q1 = 3.0
q3 = 7.0
IQR = 4.0
upperBound = 13.0
lowerBound = 3.0
skewed data = 3489
```

[31]:
```
#capping
data['length_of_service'] =[upperBound if x>upperBound else x for x in data['length_of_service']]
```

handling categorical data

[32]:
```
data['education'] = data['education'].replace(("Below Secondary","Bachelor's","Master's & above"),(1,2,3))
```

[33]:
```
from sklearn.preprocessing import LabelEncoder
```

[34]:
```
le = LabelEncoder()
```

[35]:
```
data['department'] =le.fit_transform(data['department'])
data['region'] =le.fit_transform(data['region'])
data['gender'] =le.fit_transform(data['gender'])
data['recruitment_channel'] =le.fit_transform(data['recruitment_channel'])
```

[36]:
```
data.head()
```

[36]:

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_w
0	65438	7	31	3	0	2	1	35	5.0	8.0	1	
1	65141	4	14	2	1	0	1	30	5.0	4.0	0	
2	7513	7	10	2	1	2	1	34	3.0	7.0	0	
3	2542	7	15	2	1	0	2	39	1.0	10.0	0	
4	48945	8	18	2	1	0	1	45	3.0	2.0	0	

handling imbalanced data

[37]:
```
y = data['is_promoted']
x =data.drop(columns =['is_promoted'],axis=1)
```

[38]:
```
x.head()
```

[38]:

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_y
0	65438	7	31	3	0	2	1	35	5.0	8.0	1	
1	65141	4	14	2	1	0	1	30	5.0	4.0	0	
2	7513	7	10	2	1	2	1	34	3.0	7.0	0	
3	2542	7	15	2	1	0	2	39	1.0	10.0	0	
4	48945	8	18	2	1	0	1	45	3.0	2.0	0	

[39]:
```
y.head()
```

[39]:
```
0    0
1    0
2    0
3    0
4    0
Name: is_promoted, dtype: int64
```

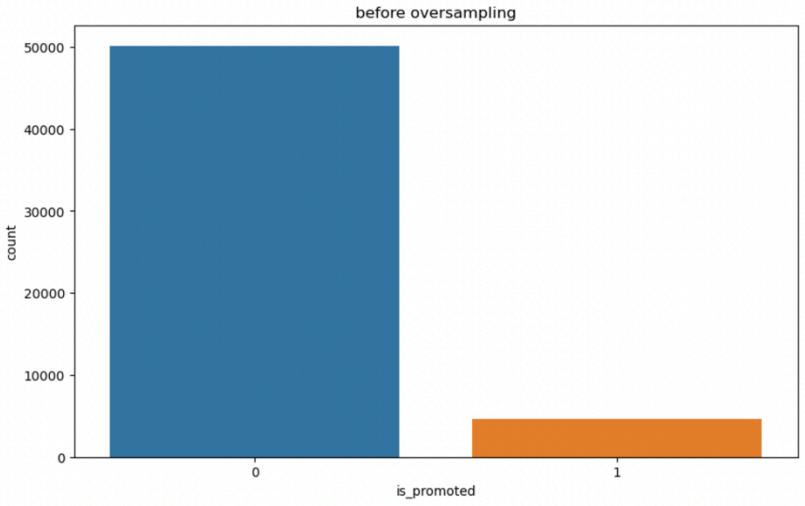
[40]:
```
from imblearn.over_sampling import SMOTE
```

[41]:
```
sm =SMOTE()
```

[42]:
```
x_resample,y_resample =sm.fit_resample(x,y)
```

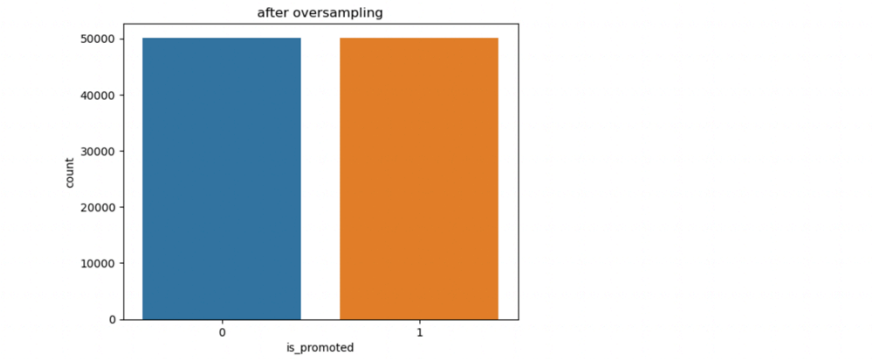
[45]:
```
plt.figure(figsize=(10,6))
sns.countplot(x=y,data=data)
plt.title('before oversampling')
```

[45]:
```
Text(0.5, 1.0, 'before oversampling')
```



```
[46]: sns.countplot(x=y_resample,data=data)
      plt.title('after oversampling')
```

```
[46]: Text(0.5, 1.0, 'after oversampling')
```



splitting data into train and test

```
[47]: from sklearn.model_selection import train_test_split
```

```
[48]: x_train,x_test,y_train,y_test = train_test_split(x_resample,y_resample,test_size=0.2,random_state=0)
```

```
[49]: x_train.shape
```

```
[49]: (80224, 13)
```

```
[50]: x_test.shape
```

```
[50]: (20056, 13)
```

```
[51]: y_train.shape
```

```
[51]: (80224,)
```

```
[52]: y_test.shape
```

```
[52]: (20056,)
```

4) Model Building -

Decision Tree Classifier -

```
[65]: #Create Model
      from sklearn.tree import DecisionTreeClassifier

[66]: df=DecisionTreeClassifier(criterion='entropy',random_state=0)

[67]: df.fit(x_train,y_train)

[67]: DecisionTreeClassifier
      DecisionTreeClassifier(criterion='entropy', random_state=0)

[68]: pred=df.predict(x_test)

[69]: pred

[69]: array([0, 0, 0, ..., 0, 0, 0])

[70]: y_test

[70]: array([0, 0, 0, ..., 0, 0, 0])
```

```
[71]: from sklearn.metrics import accuracy_score,confusion_matrix,classification_report

[72]: accuracy=accuracy_score(y_test,pred)
      conmat=confusion_matrix(y_test,pred)

[73]: print(accuracy)
      print(conmat)

      0.9285714285714286
      [[9819  210]
       [ 573  360]]

[74]: print(classification_report(y_test,pred))

                   precision    recall  f1-score   support

                0       0.94      0.98      0.96     10029
                1       0.63      0.39      0.48       933

         accuracy                           0.93     10962
        macro avg       0.79      0.68      0.72     10962
     weighted avg       0.92      0.93      0.92     10962
```

Random Forest Classifier –

```
[75]: from sklearn.ensemble import RandomForestClassifier

[76]: rf=RandomForestClassifier(n_estimators=10,criterion='entropy',random_state=0)

[77]: rf.fit(x_train,y_train)

[77]: RandomForestClassifier
      RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)

[78]: pred=rf.predict(x_test)

[79]: pred

[79]: array([0, 0, 0, ..., 0, 0, 0])

[80]: y_test

[80]: array([0, 0, 0, ..., 0, 0, 0])
```

KNN Model –

```
[90]: from sklearn.neighbors import KNeighborsClassifier

[91]: knn=KNeighborsClassifier()

[92]: knn.fit(x_train,y_train)

[92]: KNeighborsClassifier
      KNeighborsClassifier()

[93]: pred=knn.predict(x_test)

[94]: pred

[94]: array([0, 0, 0, ..., 0, 0, 0])

[95]: y_test

[95]: array([0, 0, 0, ..., 0, 0, 0])
```

XgBoost Model –

```
[100]: import xgboost as xgb

[101]: xg=xgb.XGBClassifier()

[102]: xg.fit(x_train,y_train)

[102]: XGBClassifier
       XGBClassifier(base_score=None, booster=None, callbacks=None,
                     colsample_bylevel=None, colsample_bynode=None,
                     colsample_bytree=None, device=None, early_stopping_rounds=None,
                     enable_categorical=False, eval_metric=None, feature_types=None,
                     gamma=None, grow_policy=None, importance_type=None,
                     interaction_constraints=None, learning_rate=None, max_bin=None,
                     max_cat_threshold=None, max_cat_to_onehot=None,
                     max_delta_step=None, max_depth=None, max_leaves=None,
                     min_child_weight=None, missing=nan, monotone_constraints=None,
                     multi_strategy=None, n_estimators=None, n_jobs=None,
                     num_parallel_tree=None, random_state=None, ...)

[103]: pred=xg.predict(x_test)

[104]: pred

[104]: array([0, 0, 0, ..., 0, 0, 0])

[105]: y_test

[105]: array([0, 0, 0, ..., 0, 0, 0])
```

Evaluating performnace and saving the model

```
[139]: from sklearn.model_selection import cross_val_score

[140]: cv =cross_val_score(rf,x_resample,y_resample,cv=5)
       np.mean(cv)

[140]: 0.9316114878340646

[141]: cv =cross_val_score(df,x_resample,y_resample,cv=5)
       np.mean(cv)

[141]: 0.9133226964499401

[142]: cv =cross_val_score(knn,x_resample,y_resample,cv=5)
       np.mean(cv)

[142]: 0.7992321499800559

[143]: cv =cross_val_score(xg,x_resample,y_resample,cv=5)
       np.mean(cv)

[143]: 0.9527024331870761

[144]: pickle.dump(df,open('promotion.pkl','wb'))
```

5) Application Building –

Python Code file (project.py) :-

```
from flask import Flask, render_template, request
import pickle

app = Flask(__name__)

# Load the model
model = pickle.load(open("promotion.pkl", "rb"))

@app.route('/')
def index():
    return render_template("predict.html")

@app.route('/home')
def home():
    return render_template("home.html")

@app.route('/about')
def about():
    return render_template("about.html")

@app.route('/submit')
def submit():
    return render_template("submit.html")

@app.route('/predict')
def predict():
    return render_template("predict.html")
```

```
@app.route('/datas', methods=["POST"])
def do():
    d = request.form["dept"]

    department_mapping = {
        "Sales & Marketing": 7,
        "Operations": 4,
        "Technology": 8,
        "Analytics": 0,
        "R&D": 6,
        "Procurement": 5,
        "Finance": 1,
        "HR": 2,
        "Legal": 3
    }
    d = department_mapping.get(d, 0)

    num_of_training = int(request.form["not"])
    pre_yr_rating = float(request.form["pyr"])
    len_of_service = int(request.form["los"])
    kpi = int(request.form["kpi"])
    award = int(request.form["aw"])
    avg_training_score = float(request.form["ats"])

    data = [[d, num_of_training, pre_yr_rating, len_of_service, kpi, award, avg_training_score]]

    p = model.predict(data)

    if p == 0:
        text = 'Sorry, you are not eligible for promotion'
    else:
        text = 'Great, you are eligible for promotion'

    return render_template("submit.html", data=text)
```

```
if __name__ == "__main__":
    app.run(debug=True)
```

6) Output Pages –