ict-employee-absenteeism-from-work

July 6, 2025

1 Employee Absenteeism

The database was created with records of absenteeism at work from July 2007 to July 2010 at a courier company in Brazil.

---

2 Aim

The exercise will address `Absenteeism` at a company during work time.

2.1 Problem

The problem is that the work environment of today is more: - Competitive - Managers set unachievable business goals - have an elevated risk of becoming unemployed

This can be lead to an increase in pressure and stress of the employee. Those factors influence employee health, which is of course indesirable.

2.2 What is Absenteeism?

Absence from work during normal working hours resulting in temporary incapacity to execute a regular working activity. - Based on what information should we predict whether an employee is expected to be absent or not? - How should we measure absenteeism?

2.3 Purpose of the business exercise:

Explore whether a person presenting certain characteristics is expected to be away from work at some point in time or not.

We want to know for how many working hours any employee could be away from work based on information like: - How far they live from their workplace. - How many children and pets they have. - Do they have higher education?

[1]:
```
!pip install -q hvplot
!pip install -q xlrd
```

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import hvplot.pandas
```

[2]:
```
data = pd.read_excel('D:/Project4/Absent_at_work.xls')
data.columns = data.columns.str.replace(" ", "_")

data.head()
```

[2]:
```
   ID  Reason_for_absence  Month_of_absence  Day_of_the_week  Seasons  \
0  11                26.0               7.0                3        1
1  36                 0.0               7.0                3        1
2   3                23.0               7.0                4        1
3   7                 7.0               7.0                5        1
4  11                23.0               7.0                5        1

   Transportation_expense  Distance_from_Residence_to_Work  Service_time  \
0                   289.0                             36.0          13.0
1                   118.0                             13.0          18.0
2                   179.0                             51.0          18.0
3                   279.0                              5.0          14.0
4                   289.0                             36.0          13.0

    Age  Work_load_Average/day_  …  Disciplinary_failure  Education  Son  \
0  33.0                239554.0  …                   0.0        1.0  2.0
1  50.0                239554.0  …                   1.0        1.0  1.0
2  38.0                239554.0  …                   0.0        1.0  0.0
3  39.0                239554.0  …                   0.0        1.0  2.0
4  33.0                239554.0  …                   0.0        1.0  2.0

   Social_drinker  Social_smoker  Pet  Weight  Height  Body_mass_index  \
0             1.0            0.0  1.0    90.0   172.0             30.0
1             1.0            0.0  0.0    98.0   178.0             31.0
2             1.0            0.0  0.0    89.0   170.0             31.0
3             1.0            1.0  0.0    68.0   168.0             24.0
4             1.0            0.0  1.0    90.0   172.0             30.0

   Absenteeism_time_in_hours
0                        4.0
1                        0.0
2                        2.0
3                        4.0
```

```
4                          2.0

[5 rows x 21 columns]
```

3 Exploratory Data Analysis (EDA)

[3]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 740 entries, 0 to 739
Data columns (total 21 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   ID                               740 non-null    int64
 1   Reason_for_absence               737 non-null    float64
 2   Month_of_absence                 739 non-null    float64
 3   Day_of_the_week                  740 non-null    int64
 4   Seasons                          740 non-null    int64
 5   Transportation_expense           733 non-null    float64
 6   Distance_from_Residence_to_Work  737 non-null    float64
 7   Service_time                     737 non-null    float64
 8   Age                              737 non-null    float64
 9   Work_load_Average/day_           730 non-null    float64
 10  Hit_target                       734 non-null    float64
 11  Disciplinary_failure             734 non-null    float64
 12  Education                        730 non-null    float64
 13  Son                              734 non-null    float64
 14  Social_drinker                   737 non-null    float64
 15  Social_smoker                    736 non-null    float64
 16  Pet                              738 non-null    float64
 17  Weight                           739 non-null    float64
 18  Height                           726 non-null    float64
 19  Body_mass_index                  709 non-null    float64
 20  Absenteeism_time_in_hours        718 non-null    float64
dtypes: float64(18), int64(3)
memory usage: 121.5 KB
```

[4]:
```python
data = data.dropna()
```

The data doesn't have any missing values.

[5]:
```python
pd.set_option("display.float_format", "{:.2f}".format)
data.describe()
```

[5]:
```
          ID  Reason_for_absence  Month_of_absence  Day_of_the_week  Seasons  \
count 639.00              639.00            639.00           639.00   639.00
```

```
mean   17.73                19.20              6.16              3.89     2.52
std    11.04                 8.49              3.34              1.43     1.09
min     1.00                 0.00              0.00              2.00     1.00
25%     8.00                13.00              3.00              3.00     2.00
50%    18.00                23.00              6.00              4.00     3.00
75%    28.00                27.00              9.00              5.00     3.00
max    36.00                28.00             12.00              6.00     4.00

       Transportation_expense  Distance_from_Residence_to_Work  Service_time  \
count                  639.00                           639.00        639.00
mean                   221.09                            29.67         12.73
std                     64.97                            14.70          4.36
min                    118.00                             5.00          1.00
25%                    179.00                            16.00          9.00
50%                    225.00                            26.00         13.00
75%                    260.00                            50.00         16.00
max                    388.00                            52.00         29.00

          Age  Work_load_Average/day_  …  Disciplinary_failure  Education  \
count 639.00                  639.00  …                639.00     639.00
mean   36.69               270782.13  …                  0.05       1.31
std     6.57                39049.26  …                  0.22       0.69
min    27.00               205917.00  …                  0.00       1.00
25%    31.00               244387.00  …                  0.00       1.00
50%    37.00               264249.00  …                  0.00       1.00
75%    40.00               284853.00  …                  0.00       1.00
max    58.00               378884.00  …                  1.00       4.00

          Son  Social_drinker  Social_smoker    Pet  Weight  Height  \
count 639.00          639.00         639.00 639.00  639.00  639.00
mean    1.02            0.57           0.07   0.74   79.31  172.14
std     1.08            0.49           0.26   1.32   12.95    6.05
min     0.00            0.00           0.00   0.00   56.00  163.00
25%     0.00            0.00           0.00   0.00   69.00  169.00
50%     1.00            1.00           0.00   0.00   83.00  170.00
75%     2.00            1.00           0.00   1.00   89.00  172.00
max     4.00            1.00           1.00   8.00  108.00  196.00

       Body_mass_index  Absenteeism_time_in_hours
count           639.00                     639.00
mean             26.77                       7.02
std               4.34                      13.81
min              19.00                       0.00
25%              24.00                       2.00
50%              25.00                       3.00
75%              31.00                       8.00
max              38.00                     120.00
```

[8 rows x 21 columns]

[6]:
```
for column in data.columns:
    print(f"===============Column: {column}==============")
    print(f"Number of unique values: {data[column].nunique()}")
    print(f"Max: {data[column].max()}")
    print(f"Min: {data[column].min()}")
```

```
===============Column: ID==============
Number of unique values: 36
Max: 36
Min: 1
===============Column: Reason_for_absence==============
Number of unique values: 27
Max: 28.0
Min: 0.0
===============Column: Month_of_absence==============
Number of unique values: 13
Max: 12.0
Min: 0.0
===============Column: Day_of_the_week==============
Number of unique values: 5
Max: 6
Min: 2
===============Column: Seasons==============
Number of unique values: 4
Max: 4
Min: 1
===============Column: Transportation_expense==============
Number of unique values: 24
Max: 388.0
Min: 118.0
===============Column: Distance_from_Residence_to_Work==============
Number of unique values: 25
Max: 52.0
Min: 5.0
===============Column: Service_time==============
Number of unique values: 18
Max: 29.0
Min: 1.0
===============Column: Age==============
Number of unique values: 22
Max: 58.0
Min: 27.0
===============Column: Work_load_Average/day_==============
Number of unique values: 38
```

```
Max: 378884.0
Min: 205917.0
===============Column: Hit_target==============
Number of unique values: 13
Max: 100.0
Min: 81.0
===============Column: Disciplinary_failure==============
Number of unique values: 2
Max: 1.0
Min: 0.0
===============Column: Education==============
Number of unique values: 4
Max: 4.0
Min: 1.0
===============Column: Son==============
Number of unique values: 5
Max: 4.0
Min: 0.0
===============Column: Social_drinker==============
Number of unique values: 2
Max: 1.0
Min: 0.0
===============Column: Social_smoker==============
Number of unique values: 2
Max: 1.0
Min: 0.0
===============Column: Pet==============
Number of unique values: 6
Max: 8.0
Min: 0.0
===============Column: Weight==============
Number of unique values: 26
Max: 108.0
Min: 56.0
===============Column: Height==============
Number of unique values: 14
Max: 196.0
Min: 163.0
===============Column: Body_mass_index==============
Number of unique values: 17
Max: 38.0
Min: 19.0
===============Column: Absenteeism_time_in_hours==============
Number of unique values: 19
Max: 120.0
Min: 0.0
```

[7]:
```
data.columns
```

[7]:
```
Index(['ID', 'Reason_for_absence', 'Month_of_absence', 'Day_of_the_week',
       'Seasons', 'Transportation_expense', 'Distance_from_Residence_to_Work',
       'Service_time', 'Age', 'Work_load_Average/day_', 'Hit_target',
       'Disciplinary_failure', 'Education', 'Son', 'Social_drinker',
       'Social_smoker', 'Pet', 'Weight', 'Height', 'Body_mass_index',
       'Absenteeism_time_in_hours'],
      dtype='object')
```

[8]:
```
data.ID.nunique()
```

[8]:
```
36
```

[9]:
```
data.ID.value_counts().hvplot.hist(bins=data.ID.nunique(), height=300,␣
 ↪width=500)
```

[9]:
```
:Histogram   [count]   (Count)
```

ID: individual identification (in this case we have 34 employees) indicates precisely who has been away during working hours. Will this information improve our analysis in any way? No, because it's only a label variable (a number that is there to distinguish the individuals from one another, not to carry any numeric information).

So we are going to drop this column

[10]:
```
data.drop('ID', axis=1, inplace=True)
```

[13]:
```
for col in cat_col:
    print(col)
```

```
Month_of_absence
Day_of_the_week
Seasons
Service_time
Hit_target
Disciplinary_failure
Education
Son
Social_drinker
Social_smoker
Pet
Height
Body_mass_index
Absenteeism_time_in_hours
```

`Reason for Absence`: We have 28 reason of absence from 0 to 28.

[12]:
```
cat_col = [col for col in data.columns if data[col].nunique() < 20]

for col in cat_col:
    globals()[col] = data[col].value_counts().hvplot.bar(height=350, width=350)
```

[14]:
```
Education
```

[14]: :Bars   [Education]   (count)

[15]:
```
cat_col[0]
```

[15]: 'Month_of_absence'

[16]:
```
Month_of_absence
```

[16]: :Bars   [Month_of_absence]   (count)

[17]:
```
day_list=data['Day_of_the_week'].tolist()
from functools import reduce

total_days = reduce((lambda x, y: x + y), day_list)
total_days
```

[17]: 2484

[18]:
```
# Visulazing the distibution of the data for every feature
data.hvplot.hist(y='Reason_for_absence', height=350, width=350)
```

[18]: :Histogram   [Reason_for_absence]   (Count)

[19]:
```
Month_of_absence
```

[19]: :Bars   [Month_of_absence]   (count)

[20]:
```
data.hvplot.hist(y='Month_of_absence', bins=12, width=350, height=350)
```

[20]: :Histogram   [Month_of_absence]   (Count)

[21]:
```
data['Day_of_the_week'].value_counts().hvplot.bar(height=350, width=350)
```

[21]: :Bars   [Day_of_the_week]   (count)

[22]:
```
data['Seasons'].value_counts().hvplot.bar(height=350, width=350)
```

[22]: :Bars   [Seasons]   (count)

[23]:
```
data.columns
```

[23]: Index(['Reason_for_absence', 'Month_of_absence', 'Day_of_the_week', 'Seasons',
       'Transportation_expense', 'Distance_from_Residence_to_Work',
       'Service_time', 'Age', 'Work_load_Average/day_', 'Hit_target',
       'Disciplinary_failure', 'Education', 'Son', 'Social_drinker',
       'Social_smoker', 'Pet', 'Weight', 'Height', 'Body_mass_index',
       'Absenteeism_time_in_hours'],
      dtype='object')

[24]:
```
print(f"{data['Absenteeism_time_in_hours'].value_counts()}")
```

```
Absenteeism_time_in_hours
8.00      178
2.00      136
3.00       99
1.00       78
4.00       52
0.00       34
16.00      17
24.00      15
40.00       7
5.00        6
80.00       3
120.00      3
32.00       3
112.00      2
64.00       2
7.00        1
56.00       1
104.00      1
48.00       1
Name: count, dtype: int64
```

[25]:
```
print(data['Absenteeism_time_in_hours'].value_counts()[0])
```

34

[26]:
```
print(data['Absenteeism_time_in_hours'].value_counts()[1])
```

78

[27]:
```
print(f"{data['Absenteeism_time_in_hours'].value_counts().iloc[0]/
 ↪data['Absenteeism_time_in_hours'].value_counts().iloc[1]}")
```

1.3088235294117647

[28]:
```
data["Reason_for_absence"].value_counts()
```

[28]:

```
Reason_for_absence
23.00    119
28.00    102
27.00     60
13.00     51
0.00      34
19.00     33
26.00     30
25.00     30
22.00     29
11.00     24
10.00     22
18.00     16
14.00     16
1.00      16
7.00      15
12.00      8
6.00       6
21.00      5
8.00       5
24.00      3
5.00       3
9.00       3
16.00      3
15.00      2
4.00       2
3.00       1
2.00       1
Name: count, dtype: int64
```

[29]:

```
data["Reason_for_absence"] = data["Reason_for_absence"].map(
{
0: "Group_1", 1: "Group_1", 2: "Group_1", 3: "Group_1",
4: "Group_1", 5: "Group_1", 6: "Group_1", 7: "Group_1",
8: "Group_1", 9: "Group_1", 10: "Group_1", 11: "Group_1",
12: "Group_1", 13: "Group_1", 14: "Group_1", 15: "Group_2",
16: "Group_2", 17: "Group_2", 17: "Group_2", 18: "Group_3",
19: "Group_3", 20: "Group_3", 21: "Group_3", 22: "Group_4",
23: "Group_4", 24: "Group_4", 25: "Group_4", 26: "Group_4",
27: "Group_4", 28: "Group_4"
}
)
# data["Reason for Absence"] = data["Reason for Absence"].astype("category").
 ↪cat.codes
data["Reason_for_absence"].value_counts()
```

```
[29]: Reason_for_absence
Group_4    373
Group_1    207
Group_3     54
Group_2      5
Name: count, dtype: int64
```

```
[30]: data_1 = pd.get_dummies(data, columns=['Reason_for_absence'])
```

```
[31]: data_1.head()
```

```
[31]:    Month_of_absence  Day_of_the_week  Seasons  Transportation_expense  \
0                7.00                3        1                  289.00
1                7.00                3        1                  118.00
2                7.00                4        1                  179.00
3                7.00                5        1                  279.00
4                7.00                5        1                  289.00

   Distance_from_Residence_to_Work  Service_time   Age  \
0                            36.00         13.00 33.00
1                            13.00         18.00 50.00
2                            51.00         18.00 38.00
3                             5.00         14.00 39.00
4                            36.00         13.00 33.00

   Work_load_Average/day_  Hit_target  Disciplinary_failure  …  \
0               239554.00       97.00                  0.00  …
1               239554.00       97.00                  1.00  …
2               239554.00       97.00                  0.00  …
3               239554.00       97.00                  0.00  …
4               239554.00       97.00                  0.00  …

   Social_smoker  Pet  Weight  Height  Body_mass_index  \
0           0.00 1.00   90.00  172.00            30.00
1           0.00 0.00   98.00  178.00            31.00
2           0.00 0.00   89.00  170.00            31.00
3           1.00 0.00   68.00  168.00            24.00
4           0.00 1.00   90.00  172.00            30.00

   Absenteeism_time_in_hours  Reason_for_absence_Group_1  \
0                       4.00                       False
1                       0.00                        True
2                       2.00                       False
3                       4.00                        True
4                       2.00                       False

   Reason_for_absence_Group_2  Reason_for_absence_Group_3  \
```

```
0                       False                        False
1                       False                        False
2                       False                        False
3                       False                        False
4                       False                        False

   Reason_for_absence_Group_4
0                        True
1                       False
2                        True
3                       False
4                        True

[5 rows x 23 columns]
```

[32]:
```
data_1.dtypes
```

[32]:
```
Month_of_absence                   float64
Day_of_the_week                      int64
Seasons                              int64
Transportation_expense             float64
Distance_from_Residence_to_Work    float64
Service_time                       float64
Age                                float64
Work_load_Average/day_             float64
Hit_target                         float64
Disciplinary_failure               float64
Education                          float64
Son                                float64
Social_drinker                     float64
Social_smoker                      float64
Pet                                float64
Weight                             float64
Height                             float64
Body_mass_index                    float64
Absenteeism_time_in_hours          float64
Reason_for_absence_Group_1            bool
Reason_for_absence_Group_2            bool
Reason_for_absence_Group_3            bool
Reason_for_absence_Group_4            bool
dtype: object
```

[33]:
```
data_1.dropna(inplace=True)
data_1.isna().sum()
```

[33]:
```
Month_of_absence                   0
Day_of_the_week                    0
```

```
Seasons                            0
Transportation_expense             0
Distance_from_Residence_to_Work    0
Service_time                       0
Age                                0
Work_load_Average/day_             0
Hit_target                         0
Disciplinary_failure               0
Education                          0
Son                                0
Social_drinker                     0
Social_smoker                      0
Pet                                0
Weight                             0
Height                             0
Body_mass_index                    0
Absenteeism_time_in_hours          0
Reason_for_absence_Group_1         0
Reason_for_absence_Group_2         0
Reason_for_absence_Group_3         0
Reason_for_absence_Group_4         0
dtype: int64
```

[34]:
```
data_1["Education"] = data_1.Education.map({1: 0, 2: 1, 3: 1, 4: 1})
```

[35]:
```
data_1.Education.value_counts()
```

[35]:
```
Education
0    522
1    117
Name: count, dtype: int64
```

[36]:
```
print(f"Null Values Sum: {data_1.Education.isna().sum()}")
```

```
Null Values Sum: 0
```

[37]:
```
data_2 = pd.get_dummies(data_1, columns=["Education"], drop_first=True)
data_2.columns
```

[37]:
```
Index(['Month_of_absence', 'Day_of_the_week', 'Seasons',
       'Transportation_expense', 'Distance_from_Residence_to_Work',
       'Service_time', 'Age', 'Work_load_Average/day_', 'Hit_target',
       'Disciplinary_failure', 'Son', 'Social_drinker', 'Social_smoker', 'Pet',
       'Weight', 'Height', 'Body_mass_index', 'Absenteeism_time_in_hours',
       'Reason_for_absence_Group_1', 'Reason_for_absence_Group_2',
       'Reason_for_absence_Group_3', 'Reason_for_absence_Group_4',
       'Education_1'],
```

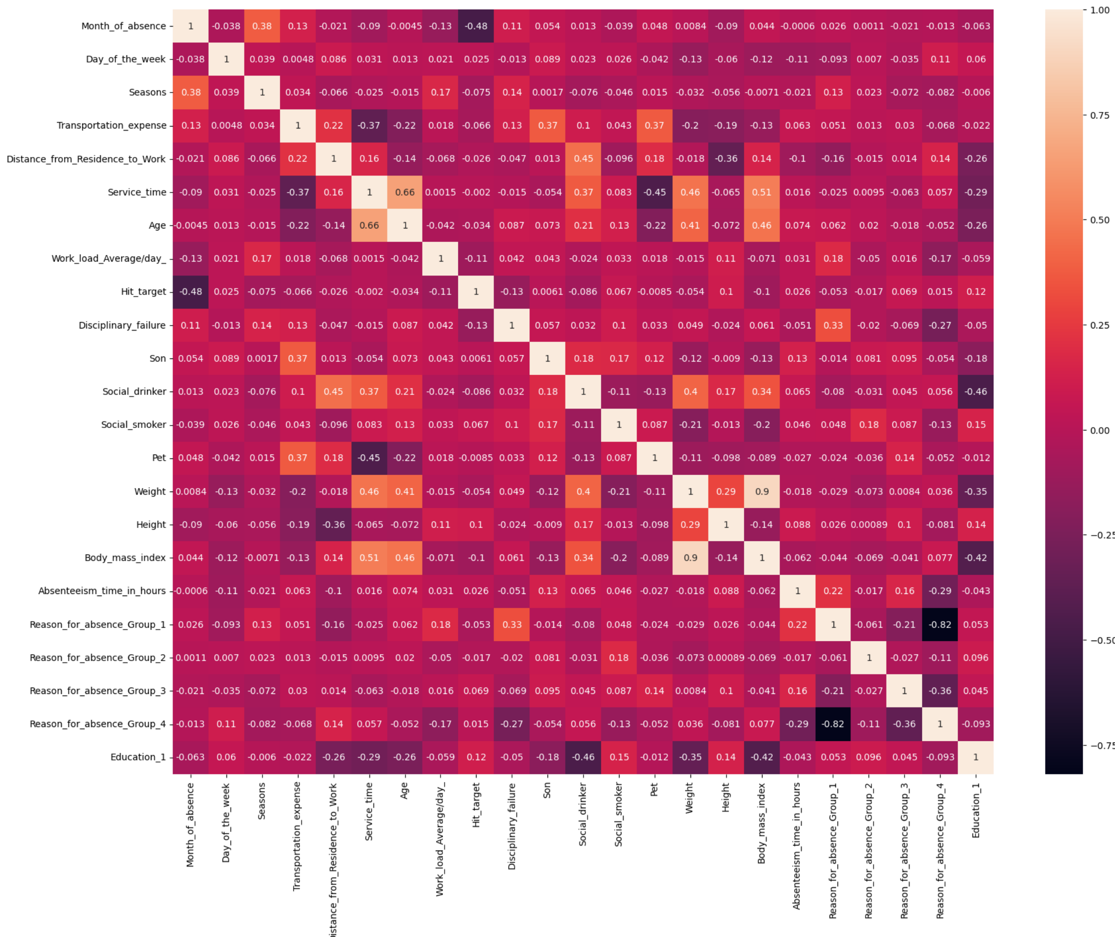
```
      dtype='object')
```

[38]:
```python
data_2.head()
```

[38]:
```
   Month_of_absence  Day_of_the_week  Seasons  Transportation_expense  \
0              7.00                3        1                  289.00
1              7.00                3        1                  118.00
2              7.00                4        1                  179.00
3              7.00                5        1                  279.00
4              7.00                5        1                  289.00

   Distance_from_Residence_to_Work  Service_time   Age  \
0                            36.00         13.00 33.00
1                            13.00         18.00 50.00
2                            51.00         18.00 38.00
3                             5.00         14.00 39.00
4                            36.00         13.00 33.00

   Work_load_Average/day_  Hit_target  Disciplinary_failure  …  Pet  Weight  \
0               239554.00       97.00                  0.00  … 1.00   90.00
1               239554.00       97.00                  1.00  … 0.00   98.00
2               239554.00       97.00                  0.00  … 0.00   89.00
3               239554.00       97.00                  0.00  … 0.00   68.00
4               239554.00       97.00                  0.00  … 1.00   90.00

   Height  Body_mass_index  Absenteeism_time_in_hours  \
0  172.00            30.00                       4.00
1  178.00            31.00                       0.00
2  170.00            31.00                       2.00
3  168.00            24.00                       4.00
4  172.00            30.00                       2.00

   Reason_for_absence_Group_1  Reason_for_absence_Group_2  \
0                       False                       False
1                        True                       False
2                       False                       False
3                        True                       False
4                       False                       False

   Reason_for_absence_Group_3  Reason_for_absence_Group_4  Education_1
0                       False                        True        False
1                       False                       False        False
2                       False                        True        False
3                       False                       False        False
4                       False                        True        False

[5 rows x 23 columns]
```

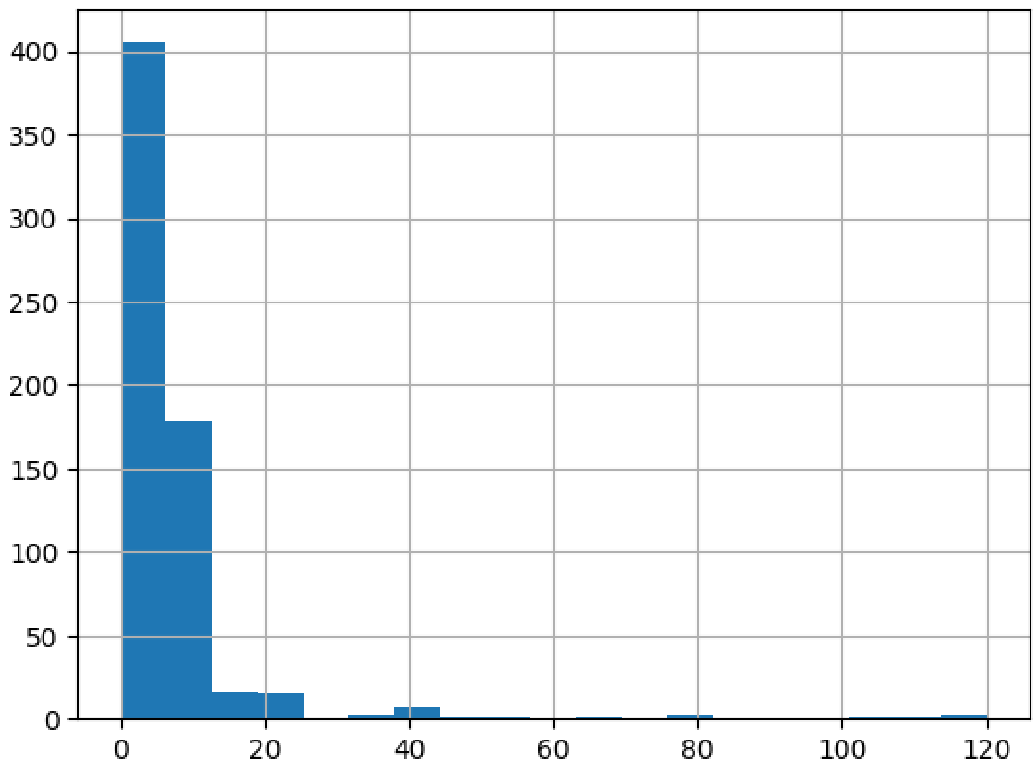
```
[39]: plt.figure(figsize=(20, 15))
      sns.heatmap(data_2.corr(), annot=True)
```

```
[39]: <Axes: >
```




```
[40]: data_2['Absenteeism_time_in_hours'].
       ↪hist(bins=data_2['Absenteeism_time_in_hours'].nunique())
```

```
[40]: <Axes: >
```



4 Applying machine learning algorthims

[41]:
```
import pickle

from sklearn.model_selection import train_test_split, cross_val_score,␣
↪GridSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.pipeline import Pipeline

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score,␣
↪classification_report
from sklearn.metrics import precision_recall_curve, roc_curve, roc_auc_score
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from sklearn.svm import SVC


X = data_2.drop('Absenteeism_time_in_hours', axis=1)
y = np.where(data_2["Absenteeism_time_in_hours"] >␣
↪data_2["Absenteeism_time_in_hours"].median(), 1, 0)
```

```
print(f'X shape: {X.shape}')
print(f'y shape: {y.shape}')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)

pipe = Pipeline([
    ('min_max_scaler', MinMaxScaler()),
    ('std_scaler', StandardScaler())
])

X_train = pipe.fit_transform(X_train)
X_test = pipe.transform(X_test)
```

```
X shape: (639, 22)
y shape: (639,)
```

[42]:
```
def evaluate(model, X_train, X_test, y_train, y_test):
    y_test_pred = model.predict(X_test)
    y_train_pred = model.predict(X_train)

    print("TRAINIG RESULTS: \n===============================")
    clf_report = pd.DataFrame(classification_report(y_train, y_train_pred,␣
↪output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_train, y_train_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_train, y_train_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")

    print("TESTING RESULTS: \n===============================")
    clf_report = pd.DataFrame(classification_report(y_test, y_test_pred,␣
↪output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_test, y_test_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_test, y_test_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")
```

5 Logistic Regression

[43]:
```
print("\n=========LOGISTIC REGRESSION============")
lr_clf = LogisticRegression(solver='liblinear', penalty='l1')
lr_clf.fit(X_train, y_train)
evaluate(lr_clf, X_train, X_test, y_train, y_test)
```

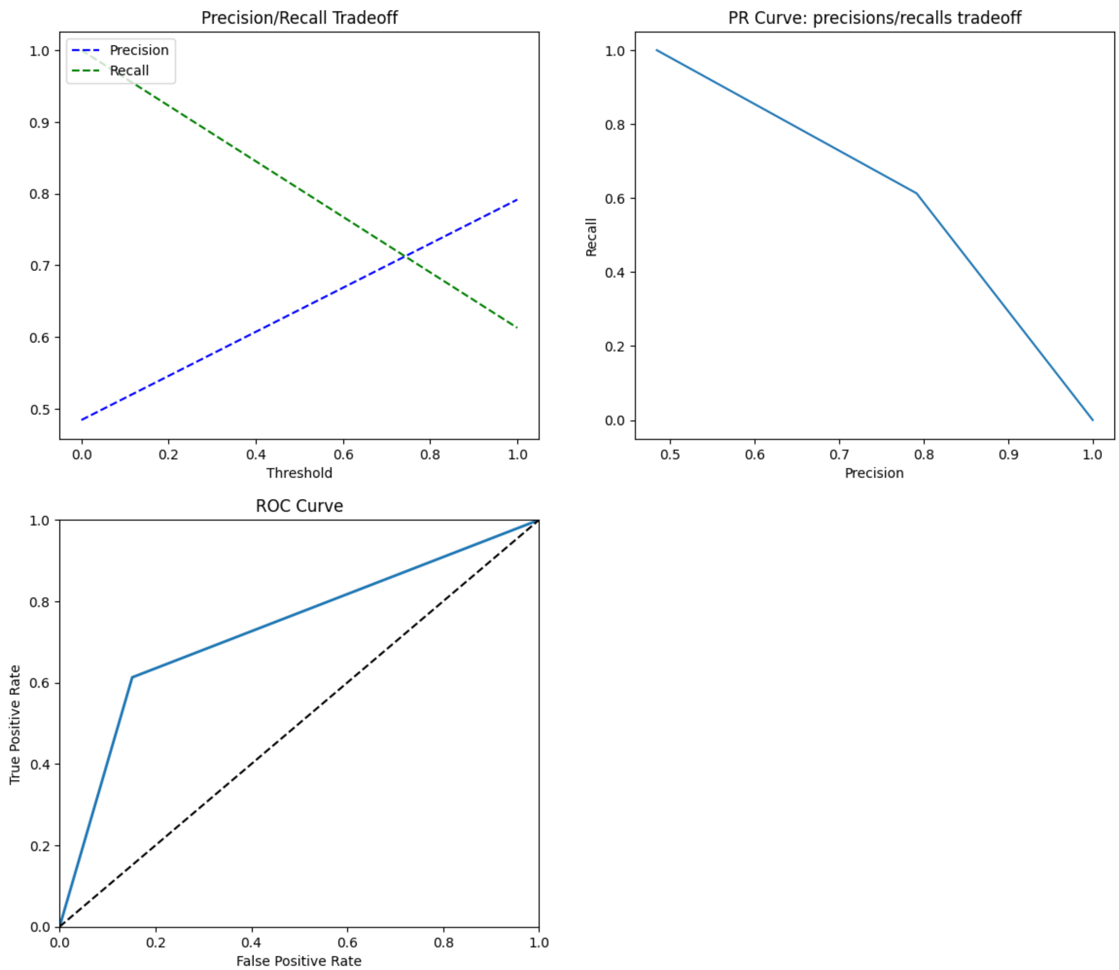
```
=========LOGISTIC REGRESSION============
TRAINIG RESULTS:
```

```
================================
CONFUSION MATRIX:
[[199  49]
 [ 54 145]]
ACCURACY SCORE:
0.7696
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision   0.79   0.75      0.77       0.77          0.77
recall      0.80   0.73      0.77       0.77          0.77
f1-score    0.79   0.74      0.77       0.77          0.77
support   248.00 199.00      0.77     447.00        447.00
TESTING RESULTS:
================================
CONFUSION MATRIX:
[[84 15]
 [36 57]]
ACCURACY SCORE:
0.7344
CLASSIFICATION REPORT:
              0     1  accuracy  macro avg  weighted avg
precision  0.70  0.79      0.73       0.75          0.74
recall     0.85  0.61      0.73       0.73          0.73
f1-score   0.77  0.69      0.73       0.73          0.73
support   99.00 93.00      0.73     192.00        192.00
```

[44]:

```
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g--", label="Recall")
    plt.xlabel("Threshold")
    plt.legend(loc="upper left")
    plt.title("Precision/Recall Tradeoff")


def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], "k--")
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')


precisions, recalls, thresholds = precision_recall_curve(y_test, lr_clf.
 ↪predict(X_test))
plt.figure(figsize=(14, 25))
plt.subplot(4, 2, 1)
```

```
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)

plt.subplot(4, 2, 2)
plt.plot(precisions, recalls)
plt.xlabel("Precision")
plt.ylabel("Recall")
plt.title("PR Curve: precisions/recalls tradeoff");

plt.subplot(4, 2, 3)
fpr, tpr, thresholds = roc_curve(y_test, lr_clf.predict(X_test))
plot_roc_curve(fpr, tpr)
```



[45]:
```
scores_dict = {
    'Logistic Regression': {
        'Train': accuracy_score(y_train, lr_clf.predict(X_train)),
        'Test': accuracy_score(y_test, lr_clf.predict(X_test)),
    },
```

```
}
```

Now you can simply select the threshold value that gives you the best precision/recall tradeoff for your task. In our case we need to predict if an employee will absent accuratly. so we need to increase

[46]:
```
scores = cross_val_score(lr_clf, X_train, y_train, cv=10)

print(f"Scores: {scores}")
print(f"Cross-Validation score mean: {scores.mean() * 100:.2f}% (+/-{scores.
↪std() * 100:.2f})")
```

```
Scores: [0.75555556 0.75555556 0.62222222 0.82222222 0.73333333 0.75555556
 0.66666667 0.79545455 0.72727273 0.77272727]
Cross-Validation score mean: 74.07% (+/-5.58)
```

[47]:
```
feature_name = X.columns
summary_table = pd.DataFrame(columns=["Features_name"], data=feature_name)
summary_table["Coefficients"] = np.transpose(lr_clf.coef_)
summary_table
```

[47]:

	Features_name	Coefficients
0	Month_of_absence	0.15
1	Day_of_the_week	-0.20
2	Seasons	-0.22
3	Transportation_expense	0.56
4	Distance_from_Residence_to_Work	-0.11
5	Service_time	-0.03
6	Age	-0.27
7	Work_load_Average/day_	0.10
8	Hit_target	-0.09
9	Disciplinary_failure	-0.98
10	Son	0.63
11	Social_drinker	0.27
12	Social_smoker	-0.01
13	Pet	-0.31
14	Weight	0.25
15	Height	-0.08
16	Body_mass_index	0.00
17	Reason_for_absence_Group_1	0.00
18	Reason_for_absence_Group_2	-0.17
19	Reason_for_absence_Group_3	0.05
20	Reason_for_absence_Group_4	-0.90
21	Education_1	0.00

[48]:
```
summary_table.index = summary_table.index + 1
summary_table.loc[0] = ['Intercept', lr_clf.intercept_[0]]
summary_table.sort_index(inplace=True)
```

[49]:
```
summary_table["Odds_ratio"] = np.exp(summary_table.Coefficients)
summary_table.sort_values(by="Odds_ratio", ascending=False)
```

[49]:

	Features_name	Coefficients	Odds_ratio
11	Son	0.63	1.87
4	Transportation_expense	0.56	1.76
12	Social_drinker	0.27	1.31
15	Weight	0.25	1.28
1	Month_of_absence	0.15	1.16
8	Work_load_Average/day_	0.10	1.11
20	Reason_for_absence_Group_3	0.05	1.05
17	Body_mass_index	0.00	1.00
18	Reason_for_absence_Group_1	0.00	1.00
22	Education_1	0.00	1.00
13	Social_smoker	-0.01	0.99
6	Service_time	-0.03	0.97
16	Height	-0.08	0.93
9	Hit_target	-0.09	0.91
5	Distance_from_Residence_to_Work	-0.11	0.90
19	Reason_for_absence_Group_2	-0.17	0.84
2	Day_of_the_week	-0.20	0.82
3	Seasons	-0.22	0.80
7	Age	-0.27	0.77
0	Intercept	-0.27	0.76
14	Pet	-0.31	0.73
21	Reason_for_absence_Group_4	-0.90	0.41
10	Disciplinary_failure	-0.98	0.38

6 Random Forest

[50]:
```
print("\n=========RANDOM FOREST============")
n_estimators = [100, 500, 1000, 1500]
max_features = ['auto', 'sqrt']
max_depth = [2, 5, 10, 15, None]
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]
criterion = ['gini', 'entropy']

params_grid = {
    'n_estimators': n_estimators,
    'max_features': max_features,
    'max_depth': max_depth,
    'min_samples_split': min_samples_split,
    'min_samples_leaf': min_samples_leaf,
    'bootstrap': bootstrap
```

```python
}

rf_clf = RandomForestClassifier(random_state=42)

rf_cv = GridSearchCV(rf_clf, params_grid, scoring="f1", cv=5, verbose=1,␣
 ↪n_jobs=-1)


rf_cv.fit(X_train, y_train)
best_params = rf_cv.best_params_
print(f"Best parameters: {best_params}")

rf_clf = RandomForestClassifier(**best_params)
rf_clf.fit(X_train, y_train)

evaluate(rf_clf, X_train, X_test, y_train, y_test)
```

```
=========RANDOM FOREST============
Fitting 5 folds for each of 720 candidates, totalling 3600 fits
```

```
C:\Users\maney\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\model_selection\_validation.py:516: FitFailedWarning:
1800 fits failed out of a total of 3600.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
866 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\maney\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\model_selection\_validation.py", line 859, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\maney\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\base.py", line 1356, in wrapper
    estimator._validate_params()
  File "C:\Users\maney\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\base.py", line 469, in _validate_params
    validate_parameter_constraints(
  File "C:\Users\maney\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\utils\_param_validation.py", line 98, in
validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features'
parameter of RandomForestClassifier must be an int in the range [1, inf), a
```

```
float in the range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got 'auto'
instead.

--------------------------------------------------------------------------------
934 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\maney\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\model_selection\_validation.py", line 859, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\maney\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\base.py", line 1356, in wrapper
    estimator._validate_params()
  File "C:\Users\maney\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\base.py", line 469, in _validate_params
    validate_parameter_constraints(
  File "C:\Users\maney\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\utils\_param_validation.py", line 98, in
validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features'
parameter of RandomForestClassifier must be an int in the range [1, inf), a
float in the range (0.0, 1.0], a str among {'sqrt', 'log2'} or None. Got 'auto'
instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\maney\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\model_selection\_search.py:1135: UserWarning: One or more of
the test scores are non-finite: [       nan        nan        nan        nan
nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.51938376 0.55300069 0.55018444 0.56937874 0.51938376 0.55300069
 0.55018444 0.56937874 0.51938376 0.55115697 0.55018444 0.5654864
 0.51938376 0.55300069 0.55018444 0.5654864  0.51938376 0.55300069
 0.55018444 0.5654864  0.51938376 0.55115697 0.55018444 0.5654864
 0.53108842 0.5582685  0.55394031 0.56937874 0.53108842 0.5582685
 0.55394031 0.56937874 0.53108842 0.56006819 0.55394031 0.56494636
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.66797106 0.67573025 0.68005013 0.67615779 0.65535434 0.67786466
 0.68131556 0.68145971 0.67628157 0.67445247 0.67961925 0.67756737
```

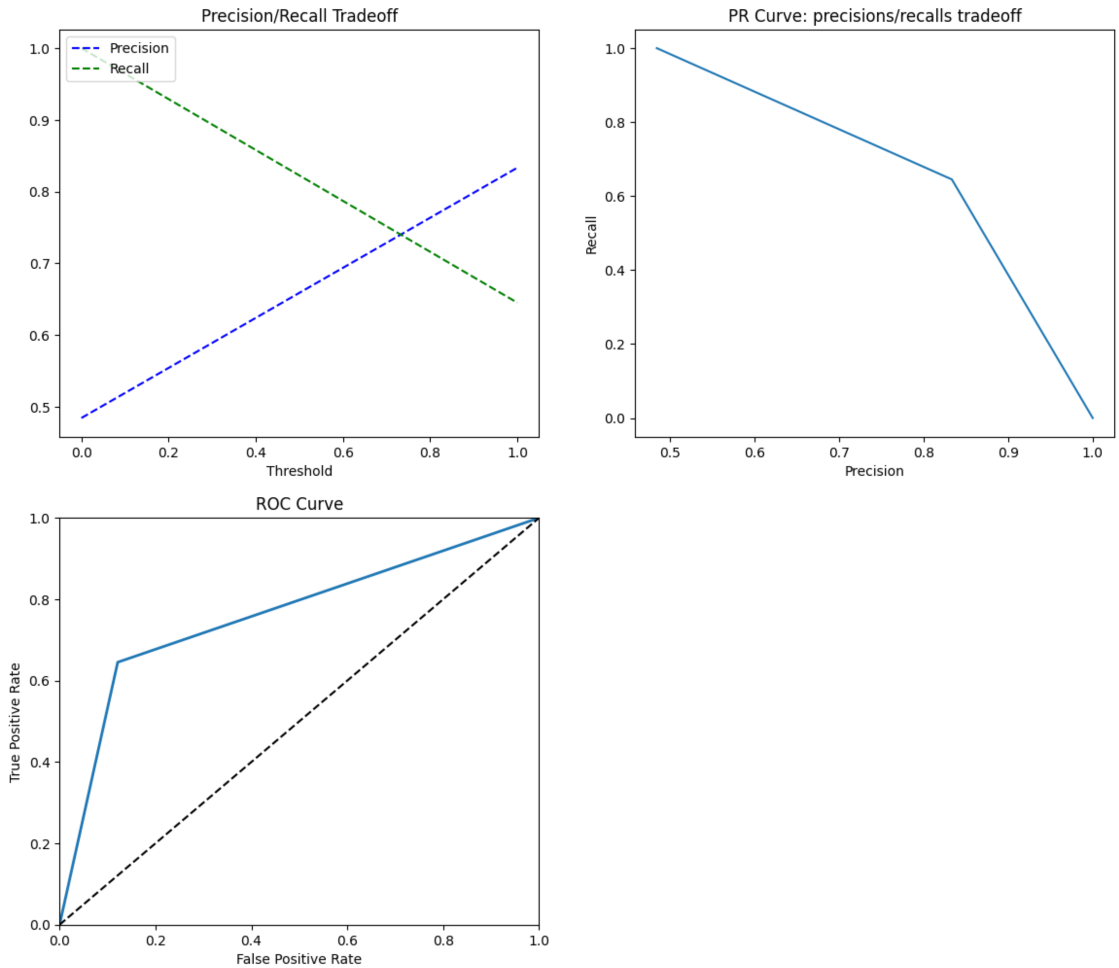
```
0.65221318 0.6699409  0.66981285 0.66686296 0.66703245 0.67226704
0.65973041 0.66515858 0.66909707 0.67229165 0.67087352 0.66928489
0.66304609 0.66475542 0.66645887 0.66169697 0.66304609 0.66475542
0.66645887 0.66169697 0.67006453 0.66888519 0.66824141 0.66292734
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.71099689 0.70631254 0.69796958 0.70252447 0.6957836  0.70831233
0.70201848 0.69700863 0.68952722 0.68766708 0.68854253 0.69313623
0.70235351 0.68543451 0.68520403 0.67690304 0.68691947 0.68017535
0.69233541 0.68611525 0.68849194 0.66255908 0.66760608 0.67663728
0.65742584 0.66522046 0.66346778 0.6750272  0.65742584 0.66522046
0.66346778 0.6750272  0.67089946 0.66222661 0.66201782 0.671172
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.68636939 0.70736318 0.70253451 0.69420599 0.70892416 0.70521724
0.70881897 0.70266667 0.69621964 0.70351826 0.69131864 0.69313623
0.6970339  0.69023033 0.69424935 0.68401462 0.69929791 0.68732558
0.68252372 0.68445025 0.69014733 0.67065835 0.67236799 0.67663728
0.65569927 0.65999919 0.66863818 0.67210139 0.65569927 0.65999919
0.66863818 0.67210139 0.66422959 0.66047393 0.66028451 0.67292468
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.68599299 0.70736318 0.70108879 0.69789177 0.70892416 0.70154898
0.70690509 0.70542779 0.69458532 0.70170067 0.68965364 0.69313623
0.69890778 0.69023033 0.6926265  0.68401462 0.68924265 0.68732558
0.68252372 0.68634499 0.69014733 0.67065835 0.66926988 0.67663728
0.65569927 0.65999919 0.66522046 0.67210139 0.65569927 0.65999919
0.66522046 0.67210139 0.66422959 0.66047393 0.66028451 0.67292468
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.55959633 0.54694873 0.56230565 0.55846879 0.55959633 0.54694873
0.56230565 0.55846879 0.55959633 0.54694873 0.56230565 0.55846879
```

0.55959633	0.54694873	0.56414937	0.55846879	0.55959633	0.54694873
0.56414937	0.55846879	0.55959633	0.54694873	0.56414937	0.55846879
0.56425903	0.55107516	0.56230565	0.55846879	0.56425903	0.55107516
0.56230565	0.55846879	0.56425903	0.55107516	0.56230565	0.55846879
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
0.68894475	0.69152438	0.69046481	0.68668211	0.69451941	0.68789295
0.68684034	0.68663659	0.68129091	0.69145982	0.69028571	0.68845923
0.68382688	0.68840328	0.68154088	0.68449864	0.68491708	0.68668211
0.6847127	0.68639337	0.67261462	0.68637323	0.68154088	0.68499965
0.67575395	0.67947855	0.68295832	0.68463899	0.67575395	0.67947855
0.68295832	0.68463899	0.67416645	0.67949478	0.67747917	0.67565723
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
0.69008445	0.69833598	0.69833598	0.69142176	0.69662894	0.69317413
0.7010302	0.70088216	0.70454817	0.69913221	0.70441496	0.70259737
0.70266185	0.70747133	0.70582736	0.70218329	0.6857241	0.69686874
0.69951539	0.69765059	0.70027418	0.6984719	0.69853511	0.7037527
0.68333	0.68302182	0.69086498	0.6880811	0.68333	0.68302182
0.69086498	0.6880811	0.69573708	0.68663698	0.6832562	0.67868003
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
0.67964593	0.70353004	0.69445901	0.69441219	0.7128794	0.69129999
0.69611579	0.68783845	0.70614801	0.69921471	0.69739712	0.69739712
0.70760644	0.69606183	0.69765673	0.69928581	0.69196944	0.69226792
0.68878198	0.69202768	0.70573612	0.69497562	0.69322123	0.69497562
0.68341684	0.68249276	0.68249276	0.68597536	0.68341684	0.68249276
0.68249276	0.68597536	0.68883916	0.68330364	0.68608753	0.67868003
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
0.69437282	0.68681577	0.69457064	0.69470734	0.70669422	0.69293045
0.6924437	0.68954732	0.70281461	0.7008866	0.69739712	0.69739712

```
 0.70279378 0.69606183 0.69407778 0.69602241 0.69011834 0.69226792
 0.68878198 0.6938048  0.70744494 0.69497562 0.69679321 0.69497562
 0.68341684 0.68249276 0.68249276 0.68597536 0.68341684 0.68249276
 0.68249276 0.68597536 0.68883916 0.68330364 0.68608753 0.67868003]
  warnings.warn(

Best parameters: {'bootstrap': False, 'max_depth': 15, 'max_features': 'sqrt',
'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100}
TRAINIG RESULTS:
===============================
CONFUSION MATRIX:
[[242   6]
 [  5 194]]
ACCURACY SCORE:
0.9754
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision   0.98   0.97      0.98       0.97          0.98
recall      0.98   0.97      0.98       0.98          0.98
f1-score    0.98   0.97      0.98       0.98          0.98
support   248.00 199.00      0.98     447.00        447.00
TESTING RESULTS:
===============================
CONFUSION MATRIX:
[[87 12]
 [33 60]]
ACCURACY SCORE:
0.7656
CLASSIFICATION REPORT:
              0     1  accuracy  macro avg  weighted avg
precision  0.72  0.83      0.77       0.78          0.78
recall     0.88  0.65      0.77       0.76          0.77
f1-score   0.79  0.73      0.77       0.76          0.76
support   99.00 93.00      0.77     192.00        192.00
```

[51]:
```
precisions, recalls, thresholds = precision_recall_curve(y_test, rf_clf.
 ↪predict(X_test))
plt.figure(figsize=(14, 25))
plt.subplot(4, 2, 1)
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)

plt.subplot(4, 2, 2)
plt.plot(precisions, recalls)
plt.xlabel("Precision")
plt.ylabel("Recall")
plt.title("PR Curve: precisions/recalls tradeoff");

plt.subplot(4, 2, 3)
```

```
fpr, tpr, thresholds = roc_curve(y_test, rf_clf.predict(X_test))
plot_roc_curve(fpr, tpr)
```



[52]:
```
scores_dict['Random Forest'] = {
        'Train': accuracy_score(y_train, rf_clf.predict(X_train)),
        'Test': accuracy_score(y_test, rf_clf.predict(X_test)),
    }
```

7 XGBoost

[53]:
```
n_estimators = [50, 100, 250]
max_depth = [2, 3, 5, 10, 15]
# booster = ['gbtree', 'gblinear']
base_score = [0.2, 0.25, 0.5, 0.75, 0.99]
learning_rate = [0.05, 0.1, 0.5, 0.9, 1]
min_child_weight = [1, 2, 3, 4]
```

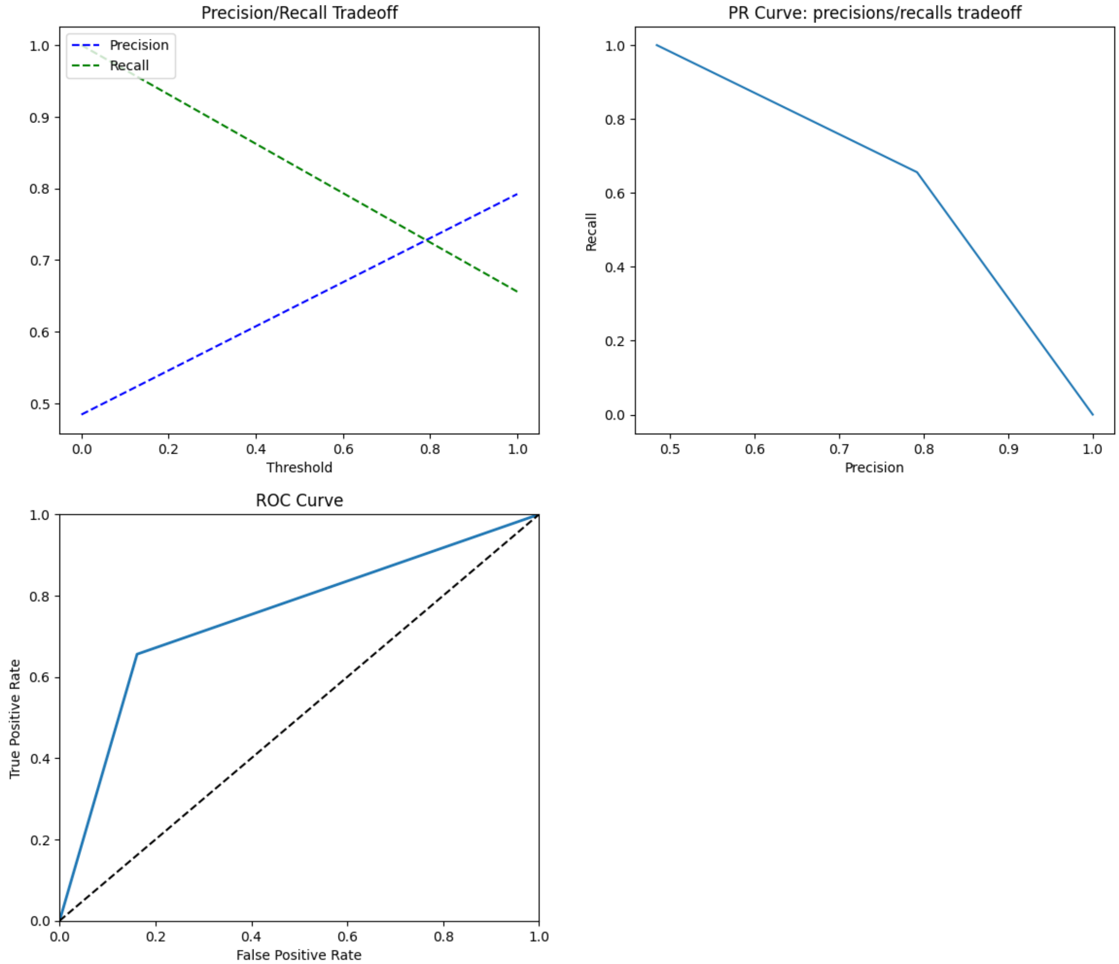
```python
subsample = [0.5, 0.75, 0.85, 0.9, 1]
colsample_bytree = [0.5, 0.75, 0.85, 0.9, 1]
colsample_bynode = [0.5, 0.75, 0.85, 0.9, 1]
colsample_bylevel = [0.5, 0.75, 0.85, 0.9, 1]

params_grid = {
    'n_estimators': n_estimators,
    'max_depth': max_depth,
    'learning_rate' : learning_rate,
    'min_child_weight' : min_child_weight,
#     'booster' : booster,
    'base_score' : base_score,
    'subsample': subsample,
#     'colsample_bytree': colsample_bytree,
#     'colsample_bynode': colsample_bynode,
#     'colsample_bylevel': colsample_bylevel,
}

xgb_clf = xgb.XGBClassifier()

xgb_cv = GridSearchCV(xgb_clf, params_grid, cv=5, scoring = 'f1',n_jobs =-1,␣
 ↪verbose=1)


xgb_cv.fit(X_train, y_train)
best_params = xgb_cv.best_params_
print(f"Best paramters: {best_params}")

xgb_clf = xgb.XGBClassifier(**best_params)
xgb_clf.fit(X_train, y_train)

evaluate(xgb_clf, X_train, X_test, y_train, y_test)
```

```
Fitting 5 folds for each of 7500 candidates, totalling 37500 fits
Best paramters: {'base_score': 0.2, 'learning_rate': 0.05, 'max_depth': 5,
'min_child_weight': 1, 'n_estimators': 250, 'subsample': 1}
TRAINIG RESULTS:
===============================
CONFUSION MATRIX:
[[243   5]
 [  9 190]]
ACCURACY SCORE:
0.9687
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision   0.96   0.97      0.97       0.97          0.97
recall      0.98   0.95      0.97       0.97          0.97
```

```
f1-score     0.97   0.96      0.97       0.97          0.97
support    248.00 199.00      0.97     447.00        447.00
TESTING RESULTS:
===============================
CONFUSION MATRIX:
[[83 16]
 [32 61]]
ACCURACY SCORE:
0.7500
CLASSIFICATION REPORT:
              0     1  accuracy  macro avg  weighted avg
precision  0.72  0.79      0.75       0.76          0.76
recall     0.84  0.66      0.75       0.75          0.75
f1-score   0.78  0.72      0.75       0.75          0.75
support   99.00 93.00      0.75     192.00        192.00
```

[54]:

```
precisions, recalls, thresholds = precision_recall_curve(y_test, xgb_clf.
 ↪predict(X_test))
plt.figure(figsize=(14, 25))
plt.subplot(4, 2, 1)
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)

plt.subplot(4, 2, 2)
plt.plot(precisions, recalls)
plt.xlabel("Precision")
plt.ylabel("Recall")
plt.title("PR Curve: precisions/recalls tradeoff");

plt.subplot(4, 2, 3)
fpr, tpr, thresholds = roc_curve(y_test, xgb_clf.predict(X_test))
plot_roc_curve(fpr, tpr)
```



[55]:
```
scores_dict['XGBoost'] = {
        'Train': accuracy_score(y_train, xgb_clf.predict(X_train)),
        'Test': accuracy_score(y_test, xgb_clf.predict(X_test)),
    }
```

8 Support Vector Machine

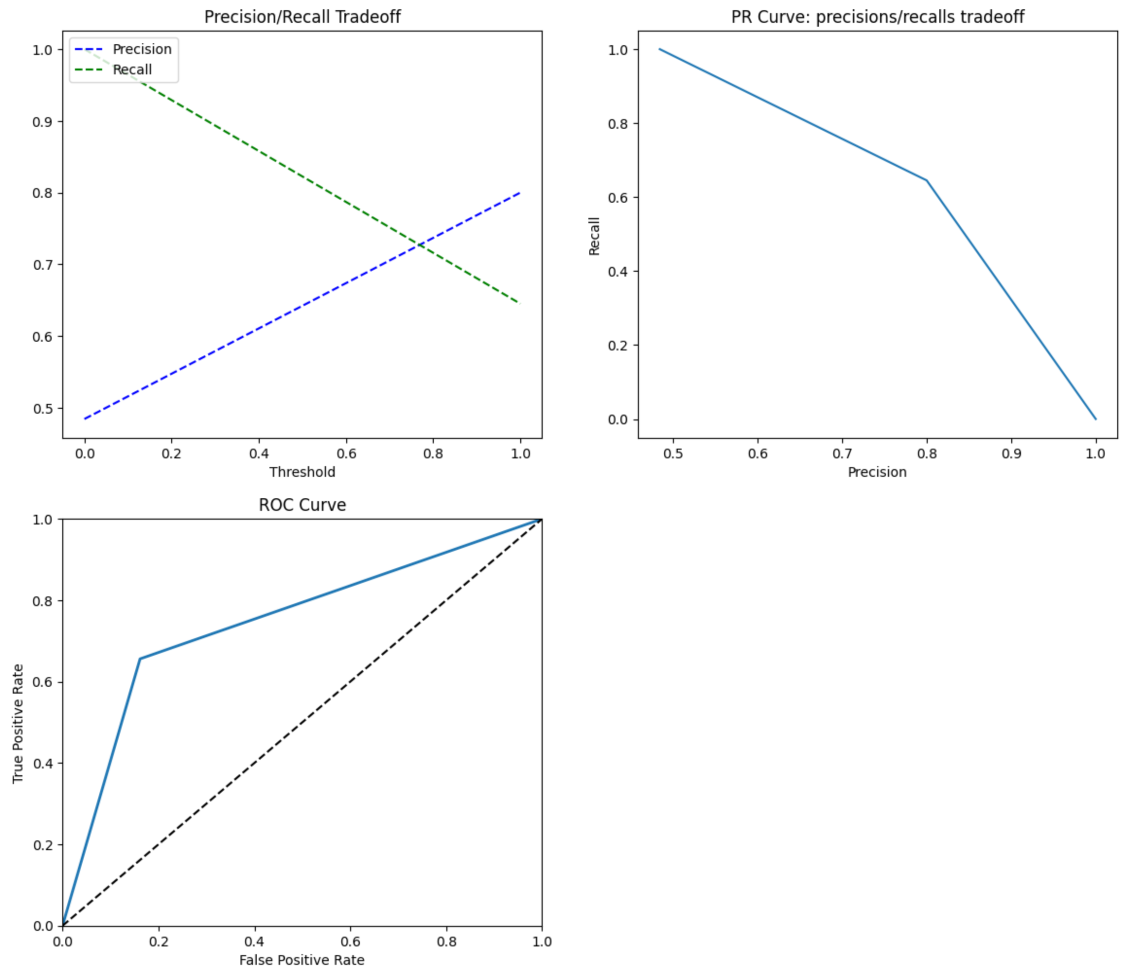
[56]:
```
param_grid = {
    'C': [60, 70, 75, 65 ],
    'gamma': [0.002, 0.001, 0.0009, 0.0008, 0.0007],
    'kernel': ['rbf', 'poly', 'linear']
}

svm_cv = GridSearchCV(SVC(), param_grid, scoring='f1', verbose=1, cv=5)
svm_cv.fit(X_train, y_train)
```

```python
best_params = svm_cv.best_params_
print(f"Best params: {best_params}")

svm_clf = SVC(**best_params)
svm_clf.fit(X_train, y_train)
evaluate(svm_clf, X_train, X_test, y_train, y_test)
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits
Best params: {'C': 60, 'gamma': 0.002, 'kernel': 'linear'}
TRAINIG RESULTS:
===============================
CONFUSION MATRIX:
[[193  55]
 [ 56 143]]
ACCURACY SCORE:
0.7517
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision   0.78   0.72      0.75       0.75          0.75
recall      0.78   0.72      0.75       0.75          0.75
f1-score    0.78   0.72      0.75       0.75          0.75
support   248.00 199.00      0.75     447.00        447.00
TESTING RESULTS:
===============================
CONFUSION MATRIX:
[[84 15]
 [33 60]]
ACCURACY SCORE:
0.7500
CLASSIFICATION REPORT:
              0     1  accuracy  macro avg  weighted avg
precision  0.72  0.80      0.75       0.76          0.76
recall     0.85  0.65      0.75       0.75          0.75
f1-score   0.78  0.71      0.75       0.75          0.75
support   99.00 93.00      0.75     192.00        192.00
```

[57]:
```python
precisions, recalls, thresholds = precision_recall_curve(y_test, svm_clf.
  ↪predict(X_test))
plt.figure(figsize=(14, 25))
plt.subplot(4, 2, 1)
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)

plt.subplot(4, 2, 2)
plt.plot(precisions, recalls)
plt.xlabel("Precision")
plt.ylabel("Recall")
plt.title("PR Curve: precisions/recalls tradeoff");
```

```
plt.subplot(4, 2, 3)
fpr, tpr, thresholds = roc_curve(y_test, xgb_clf.predict(X_test))
plot_roc_curve(fpr, tpr)
```



[58]:
```
scores_dict['Support Vector Machine'] = {
        'Train': accuracy_score(y_train, svm_clf.predict(X_train)),
        'Test': accuracy_score(y_test, svm_clf.predict(X_test)),
    }
```

9 Comparing Machine Learning models

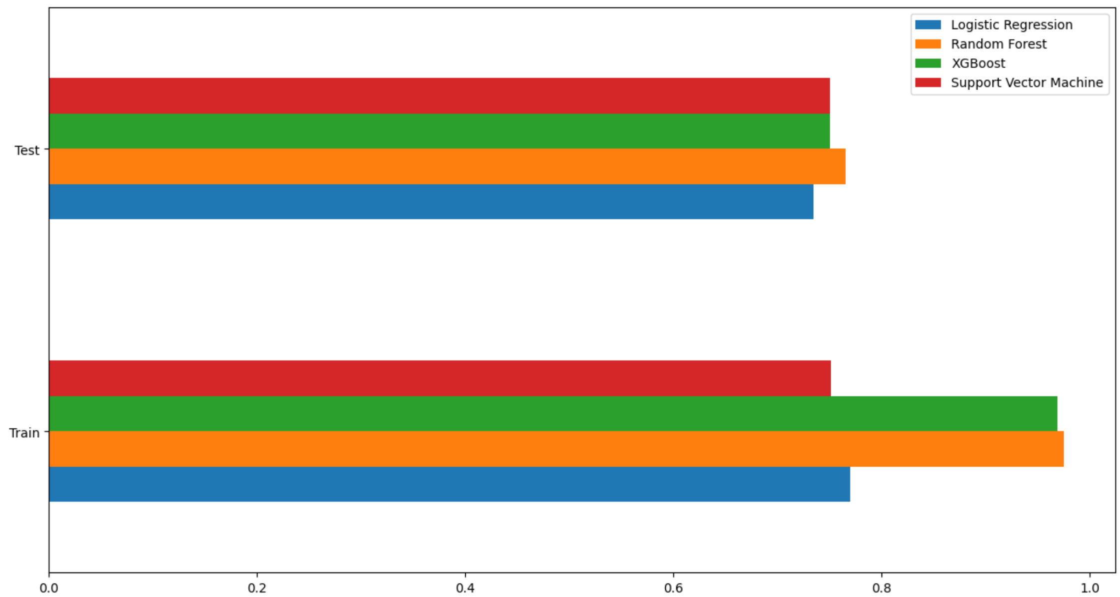
Area Under the Curve score (AUC) is good way to compare classifiers. A perfect classifier AUC will have a ROC AUC equal to 1.

[59]:
```
ml_models = {
    'Logistic Regression': lr_clf,
    'Random Forest': rf_clf,
    'XGboost': xgb_clf,
    'Support Vector Machine': svm_clf
}
for model in ml_models:
    print(f"{model.upper()} roc_auc_score: {roc_auc_score(y_test,␣
↪ml_models[model].predict(X_test)):.3f}")
```

```
LOGISTIC REGRESSION roc_auc_score: 0.731
RANDOM FOREST roc_auc_score: 0.762
XGBOOST roc_auc_score: 0.747
SUPPORT VECTOR MACHINE roc_auc_score: 0.747
```

[60]:
```
scores_df = pd.DataFrame(scores_dict)

scores_df.plot(kind='barh', figsize=(15, 8))
```

[60]: <Axes: >



10 Save the models

[61]:
```
with open('Xgb_clf', 'wb') as file:
    pickle.dump(xgb_clf, file)

with open('Rf_clf', 'wb') as file:
```

```
    pickle.dump(rf_clf, file)
```