Project: Predicting Health Treatment in Tech Company

Submitted By

NAVEEN VERMA (EBEON0522601833)

ABSTRACT

Health plays significant role in living organisms. Diagnosis and prediction of Health related diseases requires more precision, perfection and correctness because a little mistake can cause fatigue problem or death of the person, there are numerous death cases related to health and their counting is increasing exponentially day by day. To deal with the problem there is essential need of prediction system for awareness about diseases. Machine learning is the branch of Artificial Intelligence(AI), it provides prestigious support in predicting any kind of event which take training from natural events. In this paper, we calculate accuracy of machine learning algorithms for predicting health disease, for this algorithms are k-nearest neighbor, Decision Tree, Logistic Regression, support vector machine(SVM),Naive Bayes Classifier, Random Forest and XGBoost by using Kaggle repository dataset. For implementation of Python programming Anaconda(jupyter) notebook is best tool, which have many type of library, header file, that make the work more accurate and precise.

Contents

Table of Contents

Chapter - 1

INTRODUCTION

Mental health includes our emotional, psychological, and social well-being. It can affect our interactions with the world, work performance and our physical health. Nowadays, mental health topic attracts more and more attentions. A positive attitude towards seeking for treatment is important for people with mental health conditions. There are many factors that may affect this attitude. This dataset includes information about attitude about mental health in the tech workplace, individual's geographic and demographic information, and supports from workplace. We can get insights about which factors would affect the attitude and how we can do to improve the situation. In the United States, approximately 70% of adults with depression are in the workforce. Employees with depression will miss an estimated 35 million workdays a year due mental illness. Those workers experiencing unresolved depression are estimated to encounter a 35% drop in their productivity, costing employers $105 billion dollars each year. So, we can predict the health treatment at their early stage by applying machine learning algorithms on this massive amount of data to extract features that we will extract from datasets. Various machine learning techniques like logistic regression, naïve bayes, support vector machine, k nearest neighbor etc.

Problem Statement :

Mental health affects your emotional, psychological and social well-being. It affects how we think, feel, and act. The impact of mental health to an organization can mean an increase of absent days from work and a decrease in productivity and engagement. So, Develop a model that can predict whether a employee seek treatment or not. Identify the key features that lead to mental health problems in tech space.

Gaps in Existing Systems :

Gaps is that in previous analysis used 7 models. And I am also used 7 but 2 models are different.

1. Previous Analysis Models.

   Logistic Regression, Decision Tree Classifier, K-Nearest Neighbor, Random Forest Classifier, Ada Boost Classifier, Gradient Boosting Classifier, XGB Classifier.

2. New Replace Analysis Models.

   Logistic Regression, Decision Tree Classifier, K-Nearest Neighbor, Random Forest Classifier, SVM (Support Vector Machine), Naives Bayes Classifier, XGBoost accuracy.

Chapter - 2

Study of Existing Systems

Our Predictor (Y, Positive or Negative Health treatment) is determined by 22 features (X):

1. age (#)
2. Gender : 1= Female, 2=Male, 3= Transgender (Ordinal)
3. self_employed : 1=Yes, 2=No (Binary)
4. family_history : 1=Yes, 2=No (Binary)
5. work_interfere : 1=Sometimes, 2=Often, 3=Rarely, 4=Never (Ordinal)
6. no_employees : 1=01-May, 2=1-5, 3=6-25, 4=26-100, 5=100-500, 6=500-1000, 7=More than 1000, 8=Jun-25 (Ordinal)
7. remote_work : 1=Yes, 2=No (Binary)
8. tech_company : 1=Yes, 2=No (Binary)
9. Benefits : 1=Yes, 2=No, 3= Don't know (Ordinal)
10. care_options : 1=Yes, 2=No, 3=Not sure (Ordinal)
11. wellness_program : 1=Yes, 2=No, 3= Don't know (Ordinal)
12. seek_help : 1=Yes, 2=No, 3= Don't know (Ordinal)
13. Anonymity : 1=Yes, 2=No, 3= Don't know (Ordinal)

14. Leave : 1=Very difficult, 2=Somewhat difficult, 3=Very easy, 4=Somewhat easy, 5=Don't know (Ordinal)
15. mental_health_consequence : 1=Yes, 2=No, 3= Maybe (Ordinal)
16. phys_health_consequence : 1=Yes, 2=No, 3= Maybe (Ordinal)
17. coworkers : 1=Yes, 2=No, 3= Some of them (Ordinal)
18. supervisor : 1=Yes, 2=No, 3= Some of them (Ordinal)
19. mental_health_interview : 1=Yes, 2=No, 3= Maybe (Ordinal)
20. phys_health_interview : 1=Yes, 2=No, 3= Maybe (Ordinal)
21. mental_vs_physical : 1=Yes, 2=No, 3= Don't know (Ordinal)
22. obs_consequence : 1=Yes, 2=No (Binary)

Note: Our data has 3 types of data:

Continuous: Which is quantitative data that can be measured

Ordinal Data: Categorical data that has a order to it (0,1,2,3,etc)

Binary Data: Data whose unit can take on only two possible states(0&1)

Chapter - 3

Methodology

Data Cleaning and Preprocessing :

The datasets which were collected from UCI machine learning repository and Kaggle website contain unfiltered data which must be filtered before the final data set can be used to train the model. Also, data has some categorical variables which must be modified into numerical values for which we used Pandas library of Python. In data cleaning step, first we checked whether there are any missing or junk values in the dataset for which we used the isnull() function. Then for handling categorical variables we converted them into numerical variables.

Machine Learning Algorithms :

a) Logistic Regression :

Logistic regression is often used a lot of times in machine learning for predicting the likelihood of response attributes when a set of explanatory independent attributes are given. It is used when the target attribute is also known as a dependent variable having categorical values like yes/no or true/false, etc. It's widely used for solving classification problems. It falls under the category of supervised machine learning. It

efficiently solves linear and binary classification problems. It is one of the most commonly used and easy to implement algorithms. It's a statistical technique to predict classes which are binary. When the target variable has two possible classes in that case it predicts the likelihood of occurrence of the event. In our dataset the target variable is categorical as it has only two classes-yes/no.

b) Random Forest :

Random Forest is the most famous and it is considered as the best algorithm for machine learning. It is a supervised learning algorithm. To achieve more accurate and consistent prediction, random forest creates several decision trees and combines them together. The major benefit of using it is its ability to solve both regression and classification issues. When building each individual tree, it employs bagging and feature randomness in order to produce an uncorrelated tree forest whose collective forecast has much better accuracy than any individual tree's prediction. Bagging enhances accuracy of machine learning methods by grouping them together. In this algorithm, during the splitting of nodes it takes only random subset of nodes into an account. When splitting a node, it looks for the best feature from a random group of features rather than the most significant feature. This results into getting better accuracy. It efficiently deals with the huge datasets. It also solves the issue of overfitting in datasets. It works as follows: First, it'll select random samples from the provided dataset. Next, for every selected sample it'll create a decision tree and it'll receive a forecasted result from every created decision tree. Then for each result which was predicted, it'll perform voting and through voting it will select the best predicted result.

c) Naive Bayes :

It is a probabilistic machine learning algorithm which is mainly used in classification problems. It is simple and easy to build. It deals with huge datasets efficiently. It can solve complicated classification problems. The existence of a specific feature in a class is assumed to be independent of the presence of any other feature according to naïve bayes theorem. It's formula is as follows : P(S|T) = P(T|S) * P(S) / P(T) Here, T is the event to be predicted, S is the class value for an event. This equation. will find out the class in which the expected feature for classification.

d) Support Vector Machine (SVM) :

It is a powerful machine learning algorithm that falls under the category of supervised learning. Many people use SVM to solve both regression and classification problems. The primary role of SVM algorithm is that it separates two classes by creating a line of hyperplanes. Data points which are closest to the hyperplane or points of the data set that, if deleted, would change the position of dividing the hyperplane are known as support vectors. As a result, they might be regarded as essential components of the data set. The margin is the distance between hyperplane and nearest data point from either collection. The goal is to select the hyperplane with the maximum possible margin between it and any point in the training set increasing the likelihood of a new data being properly classified. SVM's main objective is to find a hyperplane in N-dimensional space which will classify all the data points. The dimension of a hyperplane is actually dependent on the quantity of input features. If input has two features in that case the hyperplane will be a line and two dimensional plane.

e) K Nearest Neighbor (KNN) :

KNN is a supervised machine learning algorithm. It assumes similar objects are nearer to one another. When the parameters are continuous in that case knn is preferred. In this algorithm it classifies objects by predicting their nearest neighbor. It's simple and easy to implement and also has high speed because of which it is preferred over the other algorithms when it comes to solving classification problems. The algorithm classifies whether or not the employee has health problem by taking the health treatment dataset as an input. It takes input parameters like age, Gender, etc. and classify person with health treatment. Algorithm takes following steps :-

Step 1: Select the value for K.

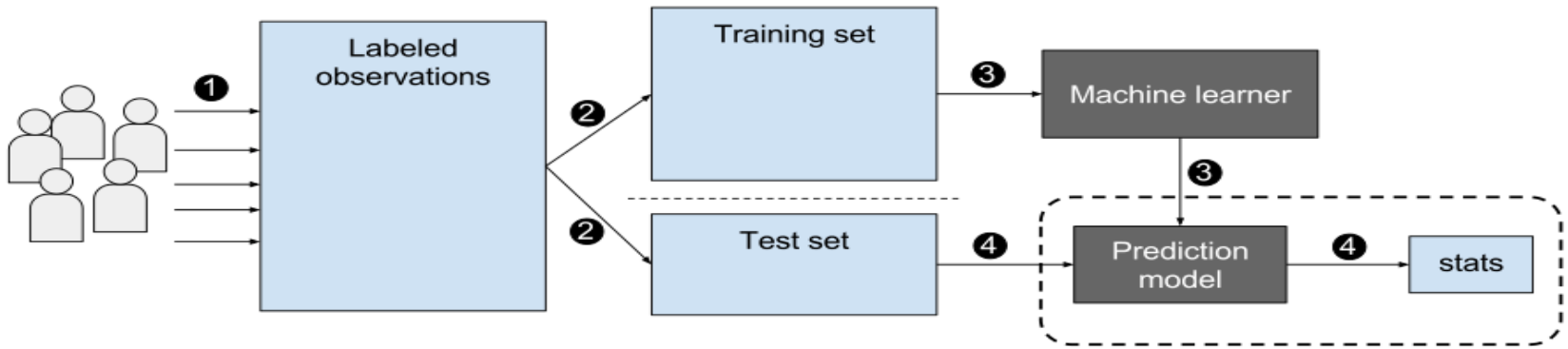
Step 2 : Find the Euclidean distance of K no. of neighbors.

Step 3 : Based on calculated distance, select the K nearest neighbors in the training data which are nearest to unknown data points.

Step 4 : Calculate no. of data points in each category among these K neighbors.

Step 5 : Assign new data points to the category which has the maximum no. of neighbors.

Step 6 : Stop.



Implementation Steps:

As we already discussed in the methodology section about some of the implementation details. So, the language used in this project is Python programming. We're running python code in anaconda navigator's Jupyter notebook. Jupyter notebook is much faster than Python IDE tools like PyCharm or Visual studio for implementing ML algorithms. The advantage of Jupyter notebook is that while writing code, it's really helpful for Data visualization and plotting some graphs like histogram and heatmap of correlated matrices. Let's revise implementation steps :

a) Dataset collection.
b) Importing Libraries : Numpy, Pandas, Scikit-learn, warnings, Matplotlib and Seaborn libraries were used.
c) Exploratory data analysis : For getting more insights about data.
d) Data cleaning and preprocessing : Checked for null and junk values using isnull() and isna().sum() functions of python. In Preprocessing phase, we did feature engineering on our dataset. As we converted categorical variables into numerical variables using function of Pandas library. Both our datasets contains some categorical variables.
e) Model selection : We first separated X's from y's. X's are features or input variables of our datasets and y's are dependent or target variables which are crucial for predicting treatment. Then using by the importing model_selection function of the sklearn library, we splitted our X's and y's into train and test split using train_test_split() function of sklearn. We splitted 70% of our data for training and 30% for testing.

f) Applied ML models and created a confusion matrix of all models.
g) Deployment of the model which gave the best accuracy.

Library

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import classification_report
```

Data Wrangling

In [2]:
```
df=pd.read_csv(r'D:\Final Project\Mental Health in Tech Company survey.csv')
df
```

Out[2]:

	Timestamp	Age	Gender	Country	state	self_employed	family_history	work_interfere	no_employees	remote_work	...	mental_health_consequence
0	27-08-2014 11:29	37	Female	United States	IL	NaN	No	Often	Jun-25	No	...	No
1	27-08-2014 11:29	44	M	United States	IN	NaN	No	Rarely	More than 1000	No	...	Maybe
2	27-08-2014 11:29	32	Male	Canada	NaN	NaN	No	Rarely	Jun-25	No	...	No
3	27-08-2014 11:29	31	Male	United Kingdom	NaN	NaN	Yes	Often	26-100	No	...	Yes
4	27-08-2014 11:30	31	Male	United States	TX	NaN	No	Never	100-500	Yes	...	No
...	...	...	...	...	...	...	...	...	...	...	...	...
1254	12-09-2015 11:17	26	male	United Kingdom	NaN	No	No	NaN	26-100	No	...	No
1255	26-09-2015	32	Male	United	IL	No	Yes	Often	26-100	Yes		No

In [5]:
```
df.shape
```

Out[5]: (1259, 27)

Number of unique value for each variable

```
In [6]:  1  df.nunique(axis=0)
```

```
Out[6]: Timestamp                    884
        Age                           53
        Gender                        49
        Country                       48
        state                         45
        self_employed                  2
        family_history                 2
        work_interfere                 4
        no_employees                   6
        remote_work                    2
        tech_company                   2
        benefits                       3
        care_options                   3
        wellness_program               3
        seek_help                      3
        anonymity                      3
        leave                          5
        mental_health_consequence      3
        phys_health_consequence        3
        coworkers                      3
        supervisor                     3
        mental_health_interview        3
        phys_health_interview          3
        mental_vs_physical             3
        obs_consequence                2
        comments                     160
        treatment                      2
        dtype: int64
```

Check null values and Dtypes

```
In [7]:  1  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1259 entries, 0 to 1258
Data columns (total 27 columns):
 #   Column                     Non-Null Count  Dtype 
---  ------                     --------------  ----- 
 0   Timestamp                  1259 non-null   object
 1   Age                        1259 non-null   int64 
 2   Gender                     1259 non-null   object
 3   Country                    1259 non-null   object
 4   state                      744 non-null    object
 5   self_employed              1241 non-null   object
 6   family_history             1259 non-null   object
 7   work_interfere             995 non-null    object
 8   no_employees               1259 non-null   object
 9   remote_work                1259 non-null   object
 10  tech_company               1259 non-null   object
 11  benefits                   1259 non-null   object
 12  care_options               1259 non-null   object
 13  wellness_program           1259 non-null   object
 14  seek_help                  1259 non-null   object
 15  anonymity                  1259 non-null   object
 16  leave                      1259 non-null   object
 17  mental_health_consequence  1259 non-null   object
 18  phys_health_consequence    1259 non-null   object
 19  coworkers                  1259 non-null   object
 20  supervisor                 1259 non-null   object
 21  mental_health_interview    1259 non-null   object
 22  phys_health_interview      1259 non-null   object
 23  mental_vs_physical         1259 non-null   object
 24  obs_consequence            1259 non-null   object
 25  comments                   164 non-null    object
 26  treatment                  1259 non-null   object
dtypes: int64(1), object(26)
memory usage: 265.7+ KB
```

Drop unnecessary columns

In [8]:
```
1 df.drop(['Timestamp','state','comments','Country'],axis=1,inplace=True)
```

- Drop Timestamp column because it's contains date, month, year and time the respondent took this questionnaire, which is irrelevant.
- Drop Country column because it's contained with the country where respondents live and 60% of answer in domination in 1 country. This column becomes irrelevant to the prediction model.
- Drop state column because it's only filled if the respondent lives in the US state, which means other respondents outside the US don't have any state ID and become missing values with 40% of data.
- Drop comments column because almost 70% of data is missing. It was an optional text box so it's reasonable to expect that many (most) respondents would leave it blank.

In [9]:
```
1 df
```

Out[9]:

	Age	Gender	self_employed	family_history	work_interfere	no_employees	remote_work	tech_company	benefits	care_options	...	leave	mental_he
0	37	Female	NaN	No	Often	Jun-25	No	Yes	Yes	Not sure	...	Somewhat easy	
1	44	M	NaN	No	Rarely	More than 1000	No	No	Don't know	No	...	Don't know	
2	32	Male	NaN	No	Rarely	Jun-25	No	Yes	No	No	...	Somewhat difficult	
3	31	Male	NaN	Yes	Often	26-100	No	Yes	No	Yes	...	Somewhat difficult	
4	31	Male	NaN	No	Never	100-500	Yes	Yes	Yes	No	...	Don't know	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1254	26	male	No	No	NaN	26-100	No	Yes	No	No	...	Somewhat easy	
1255	32	Male	No	Yes	Often	26-100	Yes	Yes	Yes	Yes	...	Somewhat difficult	

Replace unnecessary age

In [10]:
```
1 df['Age'].unique()
```

Out[10]:
```
array([         37,          44,          32,          31,          33,
                35,          39,          42,          23,          29,
                36,          27,          46,          41,          34,
                30,          40,          38,          50,          24,
                18,          28,          26,          22,          19,
                25,          45,          21,         -29,          43,
                56,          60,          54,         329,          55,
       99999999999,          48,          20,          57,          58,
                47,          62,          51,          65,          49,
             -1726,           5,          53,          61,           8,
                11,          -1,          72], dtype=int64)
```

Replace unnecessary age

In [11]:
```
1 df['Age'].replace([df['Age'][df['Age'] < 15]], np.nan, inplace = True)
2 df['Age'].replace([df['Age'][df['Age'] > 100]], np.nan,inplace = True)
```

In [12]:
```
1 df['Age'].unique()
```

Out[12]:
```
array([37., 44., 32., 31., 33., 35., 39., 42., 23., 29., 36., 27., 46.,
       41., 34., 30., 40., 38., 50., 24., 18., 28., 26., 22., 19., 25.,
       45., 21., nan, 43., 56., 60., 54., 55., 48., 20., 57., 58., 47.,
       62., 51., 65., 49., 53., 61., 72.])
```

Sum of all null values and fill

In [13]:
```
df.isna().sum()
```

Out[13]:
```
Age                          8
Gender                       0
self_employed               18
family_history               0
work_interfere             264
no_employees                 0
remote_work                  0
tech_company                 0
benefits                     0
care_options                 0
wellness_program             0
seek_help                    0
anonymity                    0
leave                        0
mental_health_consequence    0
phys_health_consequence      0
coworkers                    0
supervisor                   0
mental_health_interview      0
```

In [14]:
```
df["self_employed"].fillna(0,inplace=True)
df["work_interfere"].fillna(0,inplace=True)
df["Age"].fillna(0,inplace=True)
```

In [15]:
```
df.isna().sum()
```

Out[15]:
```
Age                0
Gender             0
self_employed      0
family_history     0
work_interfere     0
no_employees       0
```

Check Duplicate value and drop

In [16]:
```
df.duplicated().sum()
```

Out[16]:
```
4
```

In [17]:
```
df.drop_duplicates(inplace=True)
df.duplicated().sum()
```

Out[17]:
```
0
```

Check unique value and rename

In [18]:
```
df['Gender'].unique()
```

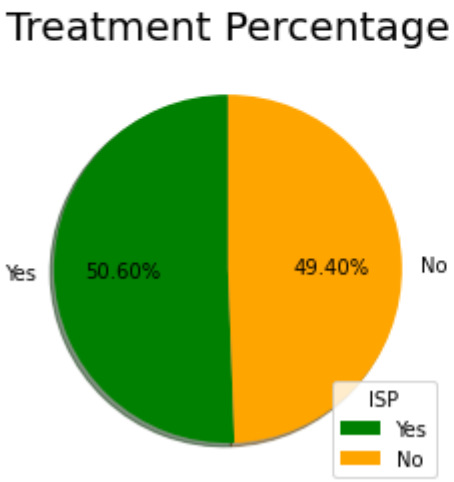
Out[18]:
```
array(['Female', 'M', 'Male', 'male', 'female', 'm', 'Male-ish', 'maile',
       'Trans-female', 'Cis Female', 'F', 'something kinda male?',
       'Cis Male', 'Woman', 'f', 'Mal', 'Male (CIS)', 'queer/she/they',
       'non-binary', 'Femake', 'woman', 'Make', 'Nah', 'All', 'Enby',
       'fluid', 'Genderqueer', 'Female ', 'Androgyne', 'Agender',
       'cis-female/femme', 'Guy (-ish) ^_^', 'male leaning androgynous',
       'Male ', 'Man', 'Trans woman', 'msle', 'Neuter', 'Female (trans)',
       'queer', 'Female (cis)', 'Mail', 'cis male', 'A little about you',
       'Malr', 'p', 'femail', 'Cis Man',
       'ostensibly male, unsure what that really means'], dtype=object)
```

In [19]:
```
df['Gender'].replace(['Female ', 'female', 'F', 'f', 'Woman', 'Female','femail', 'Cis Female', 'cis-female/femme', 'Femake',
                      'Female (cis)','woman'],'Female',inplace=True)

df['Gender'].replace(['Male ', 'male', 'M', 'm', 'Male', 'Cis Male','Man', 'cis male', 'Mail', 'Male-ish', 'Male (CIS)',
                      'Cis Man', 'msle', 'Malr', 'Mal', 'maile', 'Make',], 'Male', inplace = True)

df["Gender"].replace(['Female (trans)', 'queer/she/they', 'non-binary','fluid', 'queer', 'Androgyne',
                      'Trans-female', 'male leaning androgynous','Agender', 'A little about you', 'Nah', 'All',
                      'ostensibly male, unsure what that really means','Genderqueer', 'Enby', 'p', 'Neuter',
                      'something kinda male?','Guy (-ish) ^_^', 'Trans woman',], 'Transgender', inplace = True)
```

Exploratory Data Analysis

Check percentage of treatment

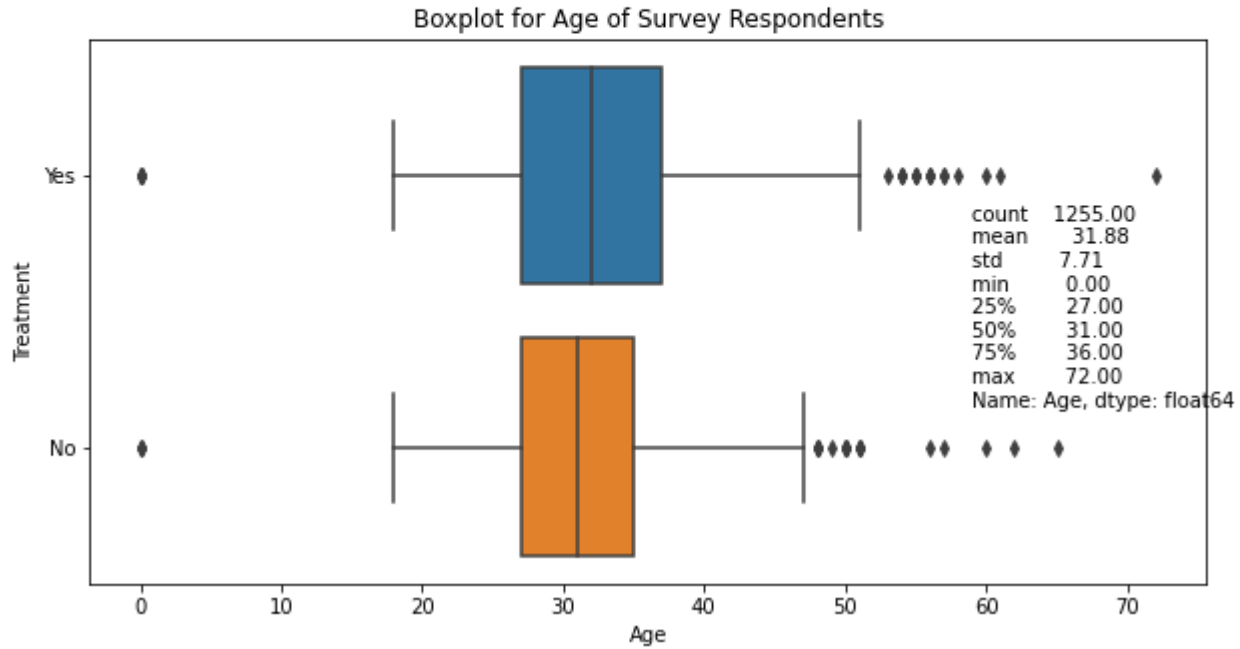
In [20]:
```
plt.title('Treatment Percentage',size=20)
count=np.array([635,620])
label=["Yes","No"]
color=['green','orange']
plt.pie(count,labels=label,autopct='%.2f%%',startangle=90,shadow=True,colors=color)
plt.legend(loc='lower right',title='ISP')
plt.show()
```



- Treatment Percentage of Yes is = 50.60%
- Treatment Percentage of No is =49.40%

BOX Plots

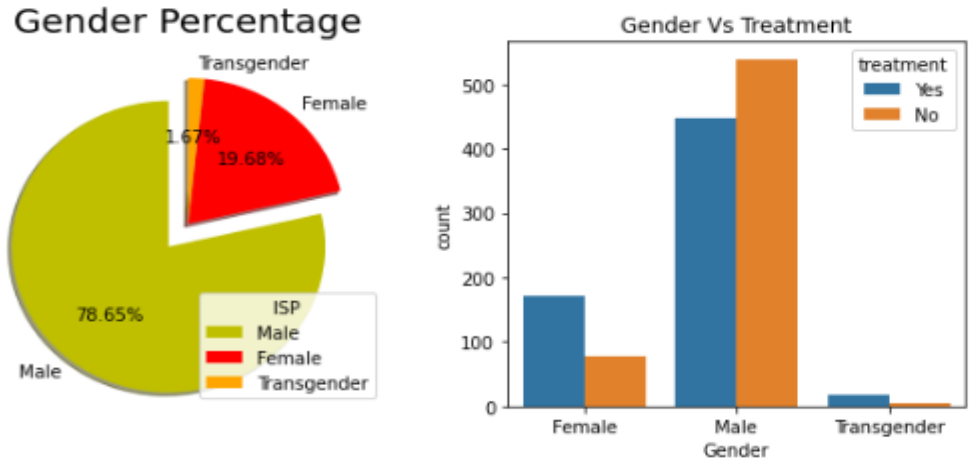
In [21]:
```
plt.figure(figsize=(10,5))
sns.boxplot(x = 'Age', y = 'treatment', data = df)
plt.title('Boxplot for Age of Survey Respondents')
age = str(df['Age'].describe().round(2))
plt.text(59, 0.85, age)
plt.ylabel('Treatment')
plt.show()
```



- From the boxplot, there is no statistically significant difference of ages between respondents that get treatment and no treatment.

Check percentage of gender and count with treatment

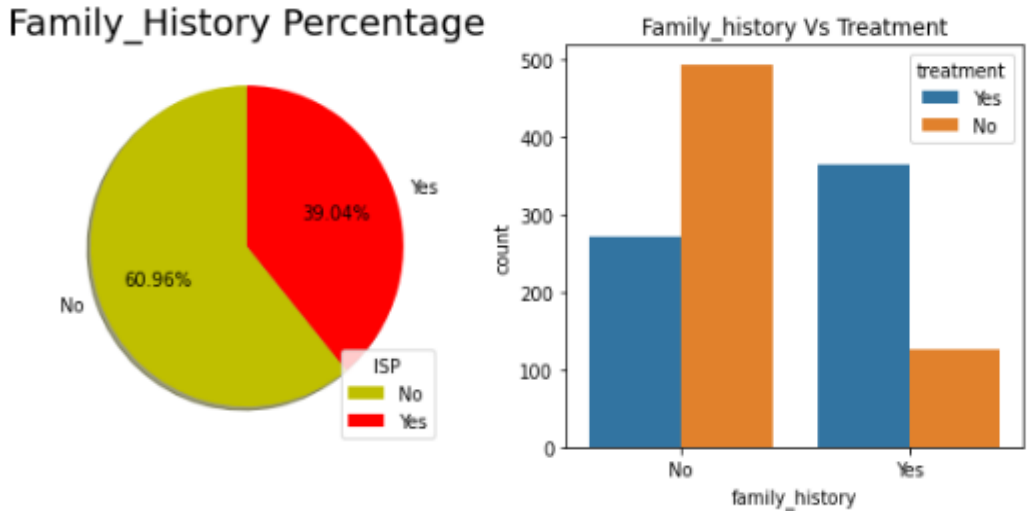
In [22]:
```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.title('Gender Percentage',size=20)
count=np.array([987,247,21])
label=['Male','Female','Transgender']
color=['y','r','orange']
plt.pie(count,labels=label,autopct='%.2f%%',startangle=90,explode=(0.2,0,0),shadow=True,colors=color)
plt.legend(loc='lower right',title='ISP')

plt.subplot(1,2,2)
plt.title('Gender Vs Treatment')
sns.countplot(df['Gender'],hue=df['treatment'])
plt.show()

```



- Almost 78.65% of respondents are male, not surprisingly, especially in the tech field. The very large gap between men and women causes higher competitive pressure for women than men. Based on the plot, female that want to get treatment is high around 70%. Maybe some of them get sexual harrassment or racism at work because female are scarce in the tech industry.

Check percentage of Family_history and count with treatment

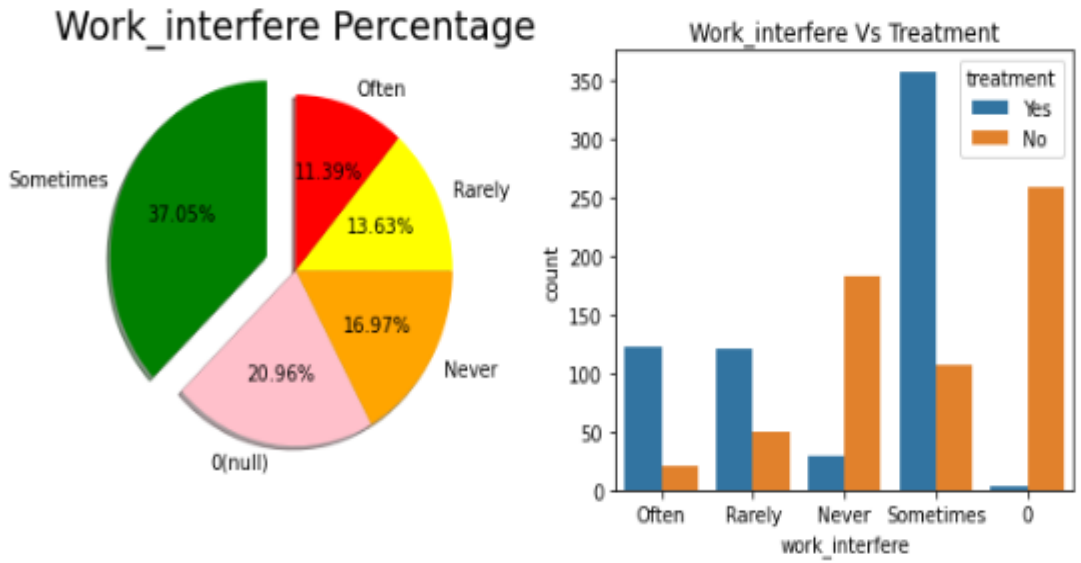
In [23]:
```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.title('Family_History Percentage',size=20)
count=np.array([765,490])
label=['No','Yes']
color=['y','r',]
plt.pie(count,labels=label,autopct='%.2f%%',startangle=90,shadow=True,colors=color)
plt.legend(loc='lower right',title='ISP')

plt.subplot(1,2,2)
plt.title('Family_history Vs Treatment')
sns.countplot(df['family_history'],hue=df['treatment'])
plt.show()
```



• From 39.04% of respondents who say that they have a family history of mental illness, the plot shows that they significantly want to get treatment rather than without a family history. This is acceptable, remember the fact that people with a family history pay more attention to mental illness. Family history is a significant risk factor for many mental health disorders.

Check percentage of Work_interfere and count with treatment

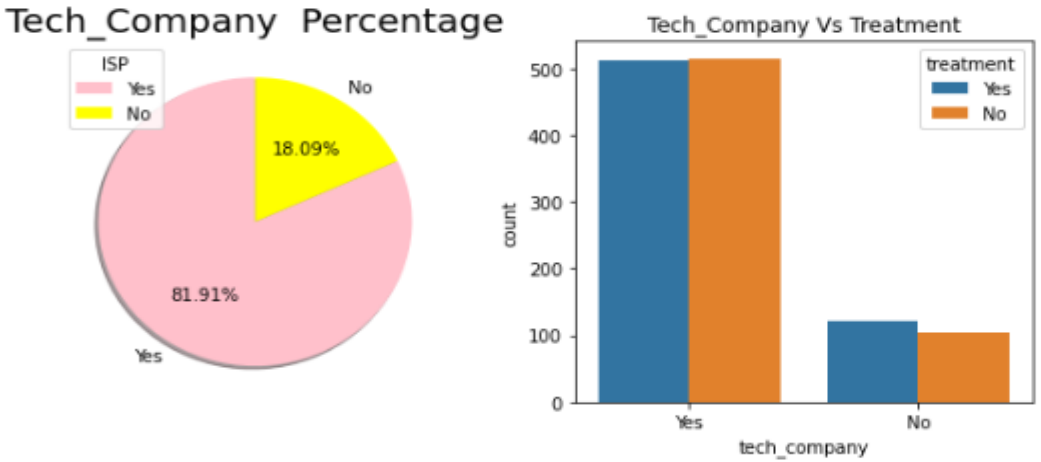
In [24]:
```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.title('Work_interfere Percentage',size=20)
count=np.array([465,263,213,171,143])
label=["Sometimes","0(null)","Never","Rarely","Often"]
color=['green','pink','orange','yellow','red']
plt.pie(count,labels=label,autopct='%.2f%%',startangle=90,shadow=True,explode=(0.2,0,0,0,0),colors=color)

plt.subplot(1,2,2)
plt.title('Work_interfere Vs Treatment')
sns.countplot(df['work_interfere'],hue=df['treatment'])
plt.show()
```



•Mental health conditions sometimes become an interfere while working about 37.05%. The plots prove that almost 80% want to get treatment.

Check percentage of Tech_company and count with treatment

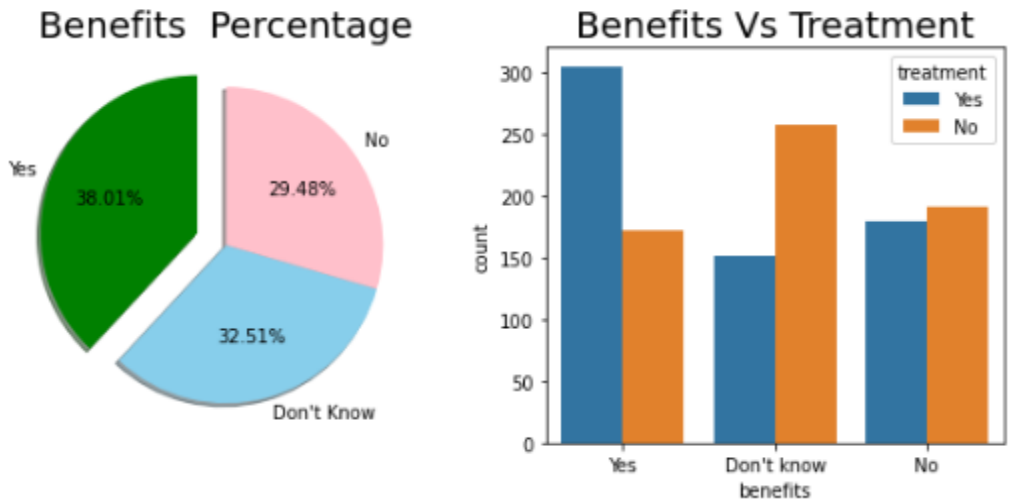
In [26]:
```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.title('Tech_Company  Percentage',size=20)
count=np.array([1028,227])
label=['Yes','No']
color=['pink','yellow']
plt.pie(count,labels=label,autopct='%.2f%%',startangle=90,shadow=True,colors=color)
plt.legend(title='ISP')

plt.subplot(1,2,2)
plt.title('Tech_Company Vs Treatment')
sns.countplot(df['tech_company'],hue=df['treatment'])
plt.show()
```



- Even the main target of the survey is the tech field, there are 18% of companies belong to the non-tech field. But it can be seen from the plot whether the company belongs to the tech field or not, mental health still becomes a big problem. I think the environment affects a lot of employees and some of them can't take it for granted like abuse at the workplace.
- However, I found that the number of employees in the technology field that want to get treatment is slightly lower than no treatment. But the non-technical field is the opposite.

Check percentage of Benefits and count with treatment

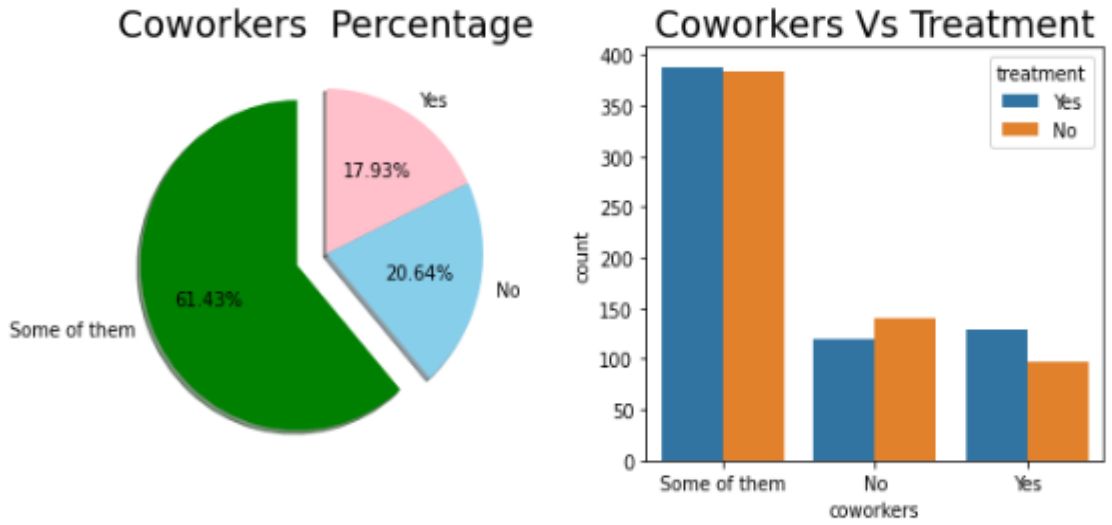
In [27]:
```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.title('Benefits  Percentage',size=20)
count=np.array([477,408,370])
label=['Yes',"Don't Know",'No']
color=['green','skyblue','pink']
plt.pie(count,labels=label,autopct='%.2f%%',startangle=90,shadow=True,explode=(0.2,0,0),colors=color)

plt.subplot(1,2,2)
plt.title('Benefits Vs Treatment',size=20)
sns.countplot(df['benefits'],hue=df['treatment'])
plt.show()
```



- Only 38% of respondents know about mental health benefits that the company provides for them.
- For employees who know the benefits, almost 60% of the employees want to get treatment. Surprisingly, there is an employee who doesn't know and says that the company doesn't provide still want to get treatment. I assume that maybe the company can't provide it properly because of budgeting or financial struggling.

Check percentage of Coworkers and count with treatment

In [28]:
```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.title('Coworkers  Percentage',size=20)
count=np.array([771,259,225])
label=['Some of them','No','Yes']
color=['green','skyblue','pink']
plt.pie(count,labels=label,autopct='%.2f%%',startangle=90,shadow=True,explode=(0.2,0,0),colors=color)

plt.subplot(1,2,2)
plt.title('Coworkers Vs Treatment',size=20)
sns.countplot(df['coworkers'],hue=df['treatment'])
plt.show()
```



- From 17% of respondents who say yes to discuss it with coworkers, 60% of them want to get treatment.
- About 61% of respondents decide to discuss some of them with coworkers. Employees who do that and want to get treatment are half of them. Let's see if the respondent will discuss it with a supervisor or not.

PreProcessing

Check dtypes

```
In [31]: 1 df.dtypes

Out[31]: Age                        float64
         Gender                      object
         self_employed               object
         family_history              object
         work_interfere              object
         no_employees                object
         remote_work                 object
         tech_company                object
         benefits                    object
         care_options                object
         wellness_program            object
         seek_help                   object
         anonymity                   object
         leave                       object
         mental_health_consequence   object
         phys_health_consequence     object
         coworkers                   object
         supervisor                  object
         mental_health_interview     object
         phys_health_interview       object
         mental_vs_physical          object
         obs_consequence             object
         treatment                   object
         dtype: object
```

• Dtypes of 22 columns are object and 1 is float. For applying machine learning convert all columns dtypes into integer.

Convert dtype object to int

In [32]:

```
df['Age']=df['Age'].astype(int)
df['Gender'].replace({'Female':1,'Male':2,'Transgender':3},inplace=True)
df['self_employed'].replace({'Yes':1,'No':2},inplace=True)
df['family_history'].replace({'Yes':1,'No':2},inplace=True)
df['treatment'].replace({'Yes':0,'No':1},inplace=True)
df['work_interfere'].replace({'Sometimes':1,'Often':2,'Rarely':3,'Never':4},inplace=True)
df['no_employees'].replace({'01-May':1,'1-5':2,'6-25':3,'26-100':4,'100-500':5,'500-1000':6,'More than 1000':7,'Jun-25':8},i
df['remote_work'].replace({'Yes':1,'No':2},inplace=True)
df['tech_company'].replace({'Yes':1,'No':2},inplace=True)
df['benefits'].replace({'Yes':1,'No':2,"Don't know":3},inplace=True)
df['care_options'].replace({'Yes':1,'No':2,"Not sure":3},inplace=True)
df['wellness_program'].replace({'Yes':1,'No':2,"Don't know":3},inplace=True)
df['seek_help'].replace({'Yes':1,'No':2,"Don't know":3},inplace=True)
df['anonymity'].replace({'Yes':1,'No':2,"Don't know":3},inplace=True)
df['leave'].replace({'Very difficult':1,'Somewhat difficult':2,'Very easy':3,'Somewhat easy':4,"Don't know":5},inplace=True)
df['mental_health_consequence'].replace({'Yes':1,'No':2,"Maybe":3},inplace=True)
df['phys_health_consequence'].replace({'Yes':1,'No':2,"Maybe":3},inplace=True)
df['coworkers'].replace({'Yes':1,'No':2,"Some of them":3},inplace=True)
df['supervisor'].replace({'Yes':1,'No':2,"Some of them":3},inplace=True)
df['mental_health_interview'].replace({'Yes':1,'No':2,"Maybe":3},inplace=True)
df['phys_health_interview'].replace({'Yes':1,'No':2,"Maybe":3},inplace=True)
df['mental_vs_physical'].replace({'Yes':1,'No':2,"Don't know":3},inplace=True)
df['obs_consequence'].replace({'Yes':1,'No':2},inplace=True)
```

Check dtypes for integer

```
In [34]:  1 df.dtypes
```

```
Out[34]: Age                          int32
         Gender                       int64
         self_employed                int64
         family_history               int64
         work_interfere               int64
         no_employees                 int64
         remote_work                  int64
         tech_company                 int64
         benefits                     int64
         care_options                 int64
         wellness_program             int64
         seek_help                    int64
         anonymity                    int64
         leave                        int64
         mental_health_consequence    int64
         phys_health_consequence      int64
         coworkers                    int64
         supervisor                   int64
         mental_health_interview      int64
         phys_health_interview        int64
         mental_vs_physical           int64
         obs_consequence              int64
         treatment                    int64
         dtype: object
```

Summarizes the count, mean, standard deviation, min and max for numeric variables

```
In [35]: 1 df.describe()
```

Out[35]:

	Age	Gender	self_employed	family_history	work_interfere	no_employees	remote_work	tech_company	benefits	care_options
count	1255.000000	1255.000000	1255.000000	1255.000000	1255.000000	1255.000000	1255.000000	1255.000000	1255.000000	1255.000000
mean	31.877291	1.819920	1.855777	1.609562	1.686056	5.443028	1.700398	1.180876	1.945020	1.897211
Std	7.709286	0.425748	0.390166	0.488043	1.388121	2.258720	0.458266	0.385069	0.838282	0.769090
min	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	27.000000	2.000000	2.000000	1.000000	1.000000	4.000000	1.000000	1.000000	1.000000	1.000000
50%	31.000000	2.000000	2.000000	2.000000	1.000000	6.000000	2.000000	1.000000	2.000000	2.000000
75%	36.000000	2.000000	2.000000	2.000000	3.000000	7.000000	2.000000	1.000000	3.000000	2.000000
.max	72.000000	3.000000	2.000000	2.000000	4.000000	8.000000	2.000000	2.000000	3.000000	3.000000

8 rows × 23 columns

Machine Learning and Predictive Analytics

- **Assign** the 22 features to X, & the last column to our classification predictor, y.

Take the X and y value

- Target: Treatment

```
In [36]: 1 x=df.drop(['treatment'],axis=1)
```

```
In [37]: 1 y=df['treatment']
```

- **Split** the data set into Training set and Test set.

```
In [38]: 1 from sklearn.model_selection import train_test_split
```

```
In [39]: 1 x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

- **Modeling/Training**
  We will now train various Classification Models on the Training set & see which yields the highest accuracy.
  We will compare the accuracy of

1. Logistic Regression
2. Decision Tree
3. Random Forest
4. K-NN (K-Nearest Neighbors)
5. SVM (Support Vector Machine)
6. Naives Bayes Classifier
7. XGBoost

Note: These are all supervised learning models.

Model 1 : Logistic Regression

In [40]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

Model1 = LogisticRegression(random_state=1)
Model1.fit(x_train,y_train)

y_pred1=Model1.predict(x_test)
print(classification_report(y_test, y_pred1))
```

```
              precision    recall  f1-score   support

           0       0.70      0.67      0.69       184
           1       0.70      0.72      0.71       193

    accuracy                           0.70       377
   macro avg       0.70      0.70      0.70       377
weighted avg       0.70      0.70      0.70       377
```

- Overall Accuracy = 70%

Model 2 : Decision Tree Classifier

In [41]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report

Model2=DecisionTreeClassifier(random_state=1)
Model2.fit(x_train,y_train)

y_pred2=Model2.predict(x_test)
print(classification_report(y_test, y_pred2))
```

```
              precision    recall  f1-score   support

           0       0.75      0.76      0.75       184
           1       0.77      0.76      0.76       193

    accuracy                           0.76       377
   macro avg       0.76      0.76      0.76       377
weighted avg       0.76      0.76      0.76       377
```

- Overall Accuracy = 76%

Model 3 : Random Forest

In [42]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

Model3=RandomForestClassifier(random_state=1)
Model3.fit(x_train,y_train)

y_pred3=Model3.predict(x_test)
print(classification_report(y_test, y_pred3))
```

```
              precision    recall  f1-score   support

           0       0.78      0.89      0.83       184
           1       0.88      0.76      0.82       193

    accuracy                           0.82       377
   macro avg       0.83      0.83      0.82       377
weighted avg       0.83      0.82      0.82       377
```

- Overall Accuracy =82%

Model 4 : K-NN (K-Nearest Neighbors)

In [43]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report

Model4=KNeighborsClassifier()
Model4.fit(x_train,y_train)

y_pred4=Model4.predict(x_test)
print(classification_report(y_test, y_pred4))
```

```
              precision    recall  f1-score   support

           0       0.63      0.69      0.66       184
           1       0.67      0.61      0.64       193

    accuracy                           0.65       377
   macro avg       0.65      0.65      0.65       377
weighted avg       0.65      0.65      0.65       377
```

- Overall Accuracy =65%

Model 5 : SVM (Support Vector Machine)

In [44]:

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report

Model5=SVC(random_state=1)
Model5.fit(x_train,y_train)

y_pred5=Model5.predict(x_test)
print(classification_report(y_test, y_pred5))
```

```
              precision    recall  f1-score   support

           0       0.63      0.65      0.64       184
           1       0.65      0.63      0.64       193

    accuracy                           0.64       377
   macro avg       0.64      0.64      0.64       377
weighted avg       0.64      0.64      0.64       377
```

- Overall Accuracy =64%

Model 6 : Naives Bayes Classifier

In [45]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report

Model6=GaussianNB()
Model6.fit(x_train,y_train)

y_pred6=Model6.predict(x_test)
print(classification_report(y_test, y_pred6))
```

```
              precision    recall  f1-score   support

           0       0.71      0.70      0.71       184
           1       0.72      0.73      0.72       193

    accuracy                           0.72       377
   macro avg       0.72      0.72      0.72       377
weighted avg       0.72      0.72      0.72       377
```

- Overall Accuracy =72%

Model 7 : XGBoost

In [46]:
```
import xgboost as xgb
from xgboost import XGBClassifier

Model7 = XGBClassifier(random_state=1)
Model7.fit(x_train, y_train)

y_pred7 = Model7.predict(x_test)
print(classification_report(y_test, y_pred7))
```

```
              precision    recall  f1-score   support

           0       0.76      0.83      0.79       184
           1       0.82      0.76      0.79       193

    accuracy                           0.79       377
   macro avg       0.79      0.79      0.79       377
weighted avg       0.79      0.79      0.79       377
```

- Overall Accuracy =79%

From comparing the 7 models, we can conclude that Model 3: Random Forest yields the highest accuracy. With an accuracy of 82%.

We have precision, recall, f1-score and support:

Precision: “How many are correctly classified among the class”

Recall: “How many of this class you find over the whole number of element of this class”

F1-score: Harmonic mean of precision and recall values. F1 score reaches its best values at 1 and worst value at 0. F1 Score = 2*((precision * recall) / precision + recall))

Support: Samples of the true response that lie in that class.

Measuring Model Performance

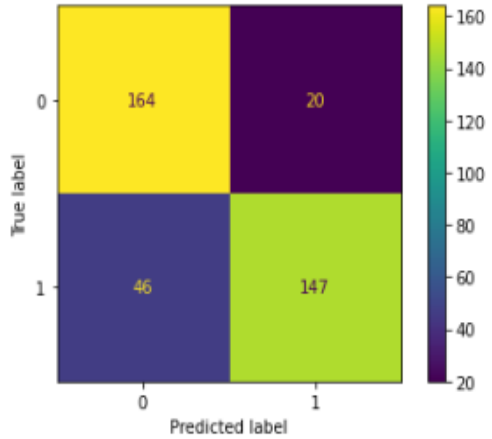
While there are other ways of measuring model performance (precision, recall, F1 Score, ROC Curve, etc), let's keep this simple and use accuracy as our metric. To do this are going to see how the model performs on new data (test set).

Accuracy is defined as: (fraction of correct predictions): correct predictions / total number of data points

Predicted Value (predicted by the test)		Actual Value (as confirmed by experiment) positives	Actual Value (as confirmed by experiment) negatives
	positives	**TP** True Positive	**FP** False Positive
	negatives	**FN** False Negative	**TN** True Negative

Confusion Matrix

In [47]:
```
1 from sklearn.metrics import plot_confusion_matrix, accuracy_score
2 plot_confusion_matrix(Model3,x_test, y_test)
3 plt.show()
4 accuracy_score(y_test, y_pred3)*100
```



Out[47]: 82.49336870026525

Confusion Matrix

- 164 is the amount of True Positives in our data, while 147 is the amount of True Negatives. 20 & 46 are the number of errors.

- There are 20 type 1 error (False Positives)- You predicted positive and it's false.

- There are 46 type 2 error (False Negatives)- You predicted negative and it's false.

- Hence if we calculate the accuracy its # Correct Predicted/ # Total. In other words, where TP, FN, FP and TN represent the number of true positives, false negatives, false positives and true negatives.

- (TP + TN)/(TP + TN + FP + FN). (164+147)/(164+147+20+46) = 0.82 = 82% accuracy

- Note: A good rule of thumb is that any accuracy above 70% is considered good, but be careful because if your accuracy is extremly high, it may be too good to be true (an example of Overfitting). Thus, 82% is the ideal accuracy.

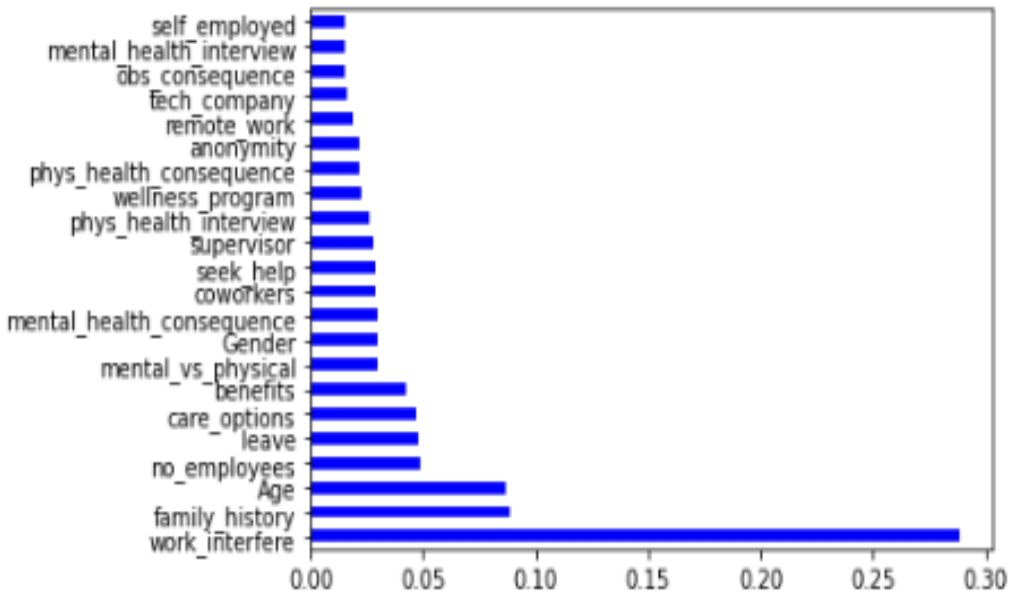
Feature Importance

- Feature Importance provides a score that indicates how helpful each feature was in our model.
- The higher the Feature Score, the more that feature is used to make key decisions & thus the more important it is.

In [48]:
```
importance = Model3.feature_importances_

for i,v in enumerate(importance):
    print('feature: %0d, Score: %.5f' % (i,v))
```

```
feature: 0, Score: 0.08702
feature: 1, Score: 0.02977
feature: 2, Score: 0.01489
feature: 3, Score: 0.08893
feature: 4, Score: 0.28849
feature: 5, Score: 0.04850
feature: 6, Score: 0.01930
feature: 7, Score: 0.01595
feature: 8, Score: 0.04282
feature: 9, Score: 0.04706
feature: 10, Score: 0.02236
feature: 11, Score: 0.02885
feature: 12, Score: 0.02153
feature: 13, Score: 0.04824
feature: 14, Score: 0.02974
feature: 15, Score: 0.02201
feature: 16, Score: 0.02909
feature: 17, Score: 0.02842
feature: 18, Score: 0.01513
feature: 19, Score: 0.02641
feature: 20, Score: 0.03000
feature: 21, Score: 0.01548
```

In [49]:
```
index= df.columns[:-1]
importance =pd.Series(Model3.feature_importances_, index=index)
importance.nlargest(23).plot(kind='barh', colormap='winter')
plt.show()
```



From the Feature Importance graph above, we can conclude that the top 4 significant features were.

- Work_interfere
- Age
- family_history
- care_options

Chapter – 4

Analysis of the Result

We used precision, F1-score, recall and accuracy evaluation metrics for evaluating our models. False Positive(FP) is when a model incorrectly predicts a positive outcome. False Negative(FN) is when a model incorrectly predicts the negative outcome. True Positive(TP) is when model correctly predicts a positive outcome. True Negative(TN) is when a model correctly predicts a negative outcome.

- Precision=TP/(TP+FP)
- Recall = TP / (TP + FN)

Machine Learning Models	Accuracy
Logistic Regression	70%
Decision Tree	76%
Random Forest	82%
KNN	65%
SVM	64%
Naive Bayes	72%
XGBoost	79%

Chapter – 5

Conclusions

- Our Random Forest algorithm yields the highest accuracy, 82%. Any accuracy above 70% is considered good, but be careful because if your accuracy is extremely high, it may be too good to be true (an example of Overfitting). Thus, 82% is the ideal accuracy.
- Out of the 22 features we examined, the top 4 significant features that helped us classify between a positive & negative Diagnosis were Work_interfere, Age, family_history, care_options.
- Our machine learning algorithm can now classify employees with health treatment. Now
- we can properly diagnose employee, & get them the help they needs to recover.

Here is access to the data set & code from my GitHub page:

https://github.com/Naveen8221/Final-Project