Final_file

September 6, 2020

1 Preprocessing of Discrete Data

[108]:
```
#importing libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

[109]:
```
# reading data
data = pd.read_csv('dataDiscrete/data.csv')
```

Data Description: This dataset contains characteristics derived from digitized imaging of fine needle aspirates of a breast tumor cell mass. The goal of this analysis is to train a machine leanring algorightms to accurately distinguish between a benign and malignant tumor to aid in clinical diagnosis.

Ten features were computed for each cell nucleus:

a) radius - ratio , real value
b) texture - ratio ,real
c) perimeter - ratio ,real
d) area -ratio , real value
e) smoothness -ratio , real value
f) compactness -ratio , real value
g) concavity -ratio , real value
h) concave points -ratio , real value
i) symmetry-ratio , real value
j) fractal dimension-ratio , real value

https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29

[110]:
```python
#shape of data
print(data.shape)
# show starting 5 rows
data.head()
```

```
(569, 33)
```

[110]:
```
         id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302         M        17.99         10.38          122.80     1001.0
1    842517         M        20.57         17.77          132.90     1326.0
2  84300903         M        19.69         21.25          130.00     1203.0
3  84348301         M        11.42         20.38           77.58      386.1
4  84358402         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   …  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0  …          17.33           184.60      2019.0            0.1622
1  …          23.41           158.80      1956.0            0.1238
2  …          25.53           152.50      1709.0            0.1444
3  …          26.50            98.87       567.7            0.2098
4  …          16.67           152.20      1575.0            0.1374

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0             0.6656           0.7119                0.2654          0.4601
1             0.1866           0.2416                0.1860          0.2750
2             0.4245           0.4504                0.2430          0.3613
3             0.8663           0.6869                0.2575          0.6638
4             0.2050           0.4000                0.1625          0.2364

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
1                  0.08902          NaN
2                  0.08758          NaN
3                  0.17300          NaN
4                  0.07678          NaN

[5 rows x 33 columns]
```

[111]:
```python
# checking for null data
data.isna().sum()
```

[111]:
```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
Unnamed: 32              569
dtype: int64
```

[112]:
```
# it shows that all the data are unique that is.
data['id'].nunique()

# here data in last column is empty and id is unique, so removing this does not␣
↪affect data

data.drop(data.columns[[-1, 0]], axis=1, inplace=True)

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
```

```
Data columns (total 31 columns):
diagnosis                  569 non-null object
radius_mean                569 non-null float64
texture_mean               569 non-null float64
perimeter_mean             569 non-null float64
area_mean                  569 non-null float64
smoothness_mean            569 non-null float64
compactness_mean           569 non-null float64
concavity_mean             569 non-null float64
concave points_mean        569 non-null float64
symmetry_mean              569 non-null float64
fractal_dimension_mean     569 non-null float64
radius_se                  569 non-null float64
texture_se                 569 non-null float64
perimeter_se               569 non-null float64
area_se                    569 non-null float64
smoothness_se              569 non-null float64
compactness_se             569 non-null float64
concavity_se               569 non-null float64
concave points_se          569 non-null float64
symmetry_se                569 non-null float64
fractal_dimension_se       569 non-null float64
radius_worst               569 non-null float64
texture_worst              569 non-null float64
perimeter_worst            569 non-null float64
area_worst                 569 non-null float64
smoothness_worst           569 non-null float64
compactness_worst          569 non-null float64
concavity_worst            569 non-null float64
concave points_worst       569 non-null float64
symmetry_worst             569 non-null float64
fractal_dimension_worst    569 non-null float64
dtypes: float64(30), object(1)
memory usage: 137.9+ KB
```

[113]:
```
# first 5 entries of dataframe
data.head()
```

[113]:
```
  diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0         M        17.99         10.38          122.80     1001.0
1         M        20.57         17.77          132.90     1326.0
2         M        19.69         21.25          130.00     1203.0
3         M        11.42         20.38           77.58      386.1
4         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
```

```
1         0.08474            0.07864          0.0869              0.07017
2         0.10960            0.15990          0.1974              0.12790
3         0.14250            0.28390          0.2414              0.10520
4         0.10030            0.13280          0.1980              0.10430

   symmetry_mean  …  radius_worst  texture_worst  perimeter_worst  \
0         0.2419  …         25.38          17.33           184.60
1         0.1812  …         24.99          23.41           158.80
2         0.2069  …         23.57          25.53           152.50
3         0.2597  …         14.91          26.50            98.87
4         0.1809  …         22.54          16.67           152.20

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0      2019.0            0.1622             0.6656           0.7119
1      1956.0            0.1238             0.1866           0.2416
2      1709.0            0.1444             0.4245           0.4504
3       567.7            0.2098             0.8663           0.6869
4      1575.0            0.1374             0.2050           0.4000

   concave points_worst  symmetry_worst  fractal_dimension_worst
0                0.2654          0.4601                  0.11890
1                0.1860          0.2750                  0.08902
2                0.2430          0.3613                  0.08758
3                0.2575          0.6638                  0.17300
4                0.1625          0.2364                  0.07678

[5 rows x 31 columns]
```

[114]:

```python
#finding out numerical and categorical

numerical_features=[feature for feature in data.columns if data[feature].dtype!
↪='O']
categorical_features=[feature for feature in  data.columns if  data[feature].
↪dtype=='O' and feature!='wage_class']

print('categorical features: ''\n',categorical_features)
print('\n')
print('numerical features: ''\n',numerical_features)
```

```
categorical features:
 ['diagnosis']


numerical features:
 ['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave points_mean',
'symmetry_mean', 'fractal_dimension_mean', 'radius_se', 'texture_se',
```

```
'perimeter_se', 'area_se', 'smoothness_se', 'compactness_se', 'concavity_se',
'concave points_se', 'symmetry_se', 'fractal_dimension_se', 'radius_worst',
'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_worst',
'compactness_worst', 'concavity_worst', 'concave points_worst',
'symmetry_worst', 'fractal_dimension_worst']
```

[115]:

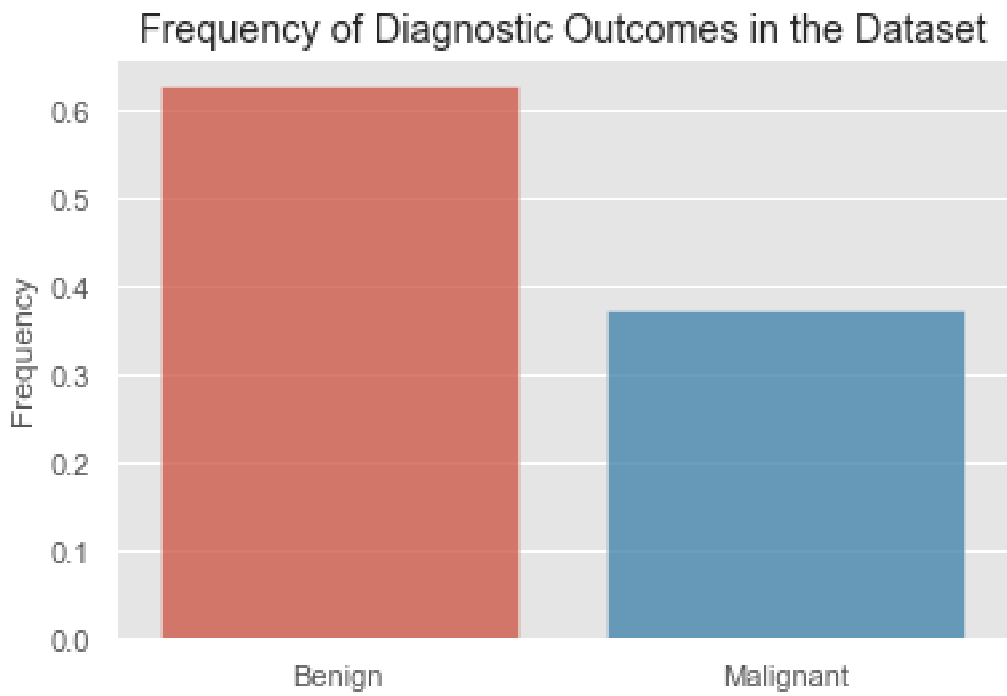
```
data_count=data.diagnosis.value_counts(normalize = True)
data_count = pd.Series(data_count)
data_count = pd.DataFrame(data_count)
data_count.index = ['Benign', 'Malignant']

data_count['Percent'] = 100*data_count['diagnosis']/sum(data_count['diagnosis'])
data_count['Percent'] = data_count['Percent'].round().astype('int')
data_count
```

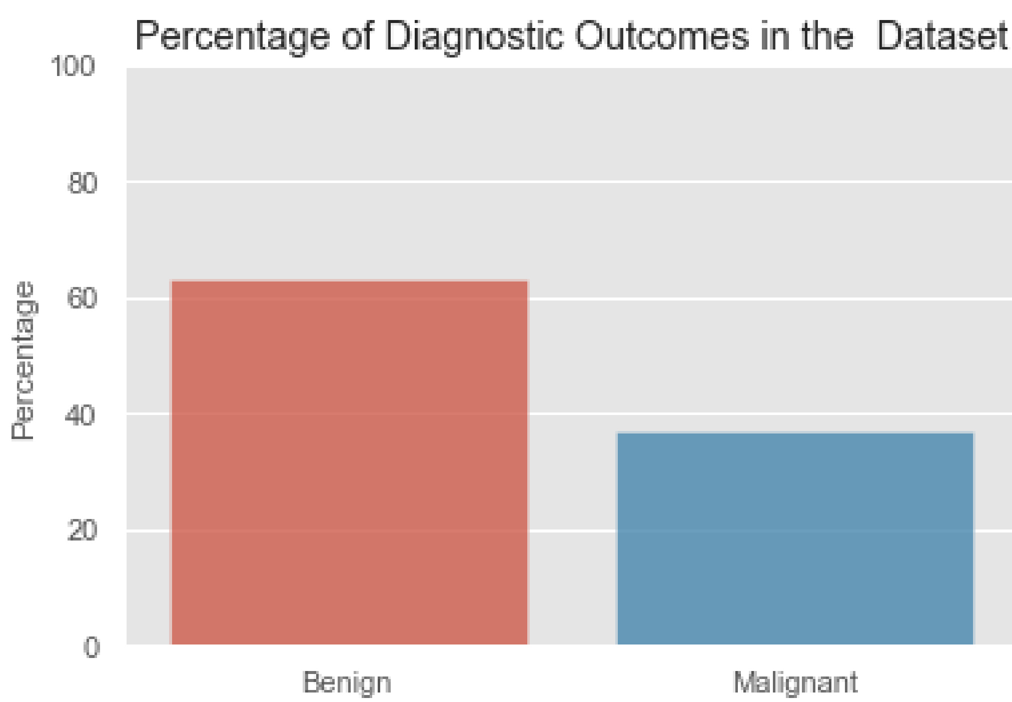
[115]:

	diagnosis	Percent
Benign	0.627417	63
Malignant	0.372583	37

[116]:

```
# Visualize frequency and percentage of Diagnostic Outcomes in the  Dataset
sns.barplot(x = ['Benign', 'Malignant'], y = 'diagnosis', data = data_count,␣
 ↪alpha = .8)
plt.title('Frequency of Diagnostic Outcomes in the Dataset')
plt.ylabel('Frequency')
plt.show()
sns.barplot(x = ['Benign', 'Malignant'], y = 'Percent', data = data_count,␣
 ↪alpha = .8)
plt.title('Percentage of Diagnostic Outcomes in the  Dataset')
plt.ylabel('Percentage')
plt.ylim(0,100)
plt.show()
```


Frequency of Diagnostic Outcomes in the Dataset
Frequency
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Benign
Malignant


Percentage of Diagnostic Outcomes in the Dataset
Percentage
0
20
40
60
80
100
Benign
Malignant

[117]:
```python
# changing categorical data to numerical
data['diagnosis']= data['diagnosis'].map({'M':1,'B':0})
```

[118]:
```python
# checking the different values contained in the diagnosis column
#Benign : 0
#Malign : 1

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

[118]:
```
0    357
1    212
Name: diagnosis, dtype: int64
```

[119]:
```python
#'duplicated()' function in pandas return the duplicate row as True and other␣
↪as False
#for counting the duplicate elements we sum all the rows
sum(data.duplicated())
```

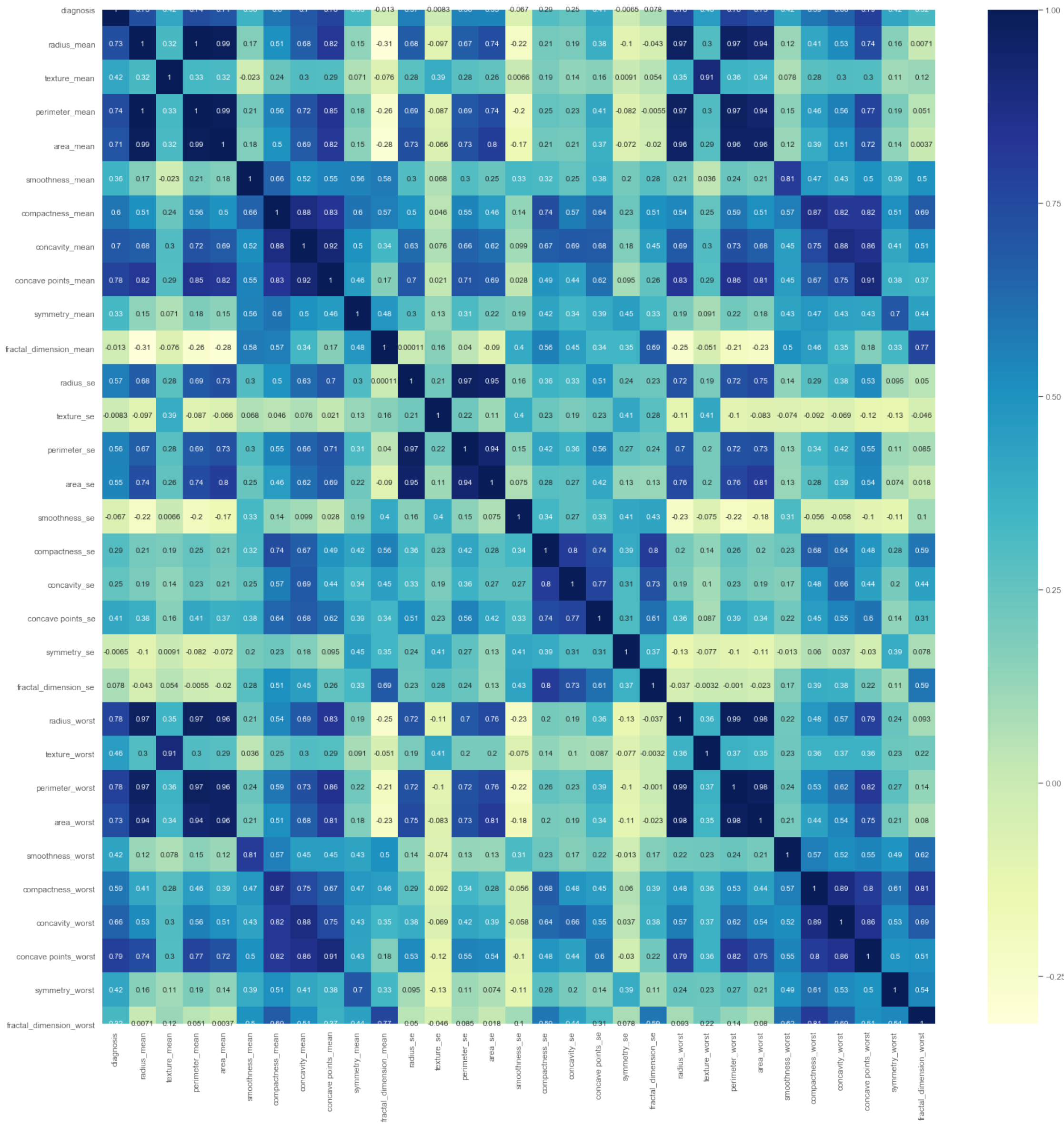
[119]:
```
0
```

[120]:
```python
# checking data shape again.
data.shape
```

[120]:
```
(569, 31)
```

[121]:
```python
# Finding correlation among features using sns' heatmap
plt.figure(figsize=(25,25))
sns.heatmap(data.corr(),annot=True, cmap="YlGnBu")
```

[121]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x299366f7dc8>
```



Observation: -> The radius, parameter and area are highly correlated as expected from their relation so from these we will use anyone of them

-> Compactness_mean, concavity_mean and concavepoint_mean are highly correlated so we will use compactness_mean from here

-> So selected Parameter for use is perimeter_mean, texture_mean, compactness_mean, symmetry_mean

Dimensionality Reduction From 30 components which are the most important ones (intrestigness)? Can we reduce our data dimension? Approach:

1. scaling

2. PCA

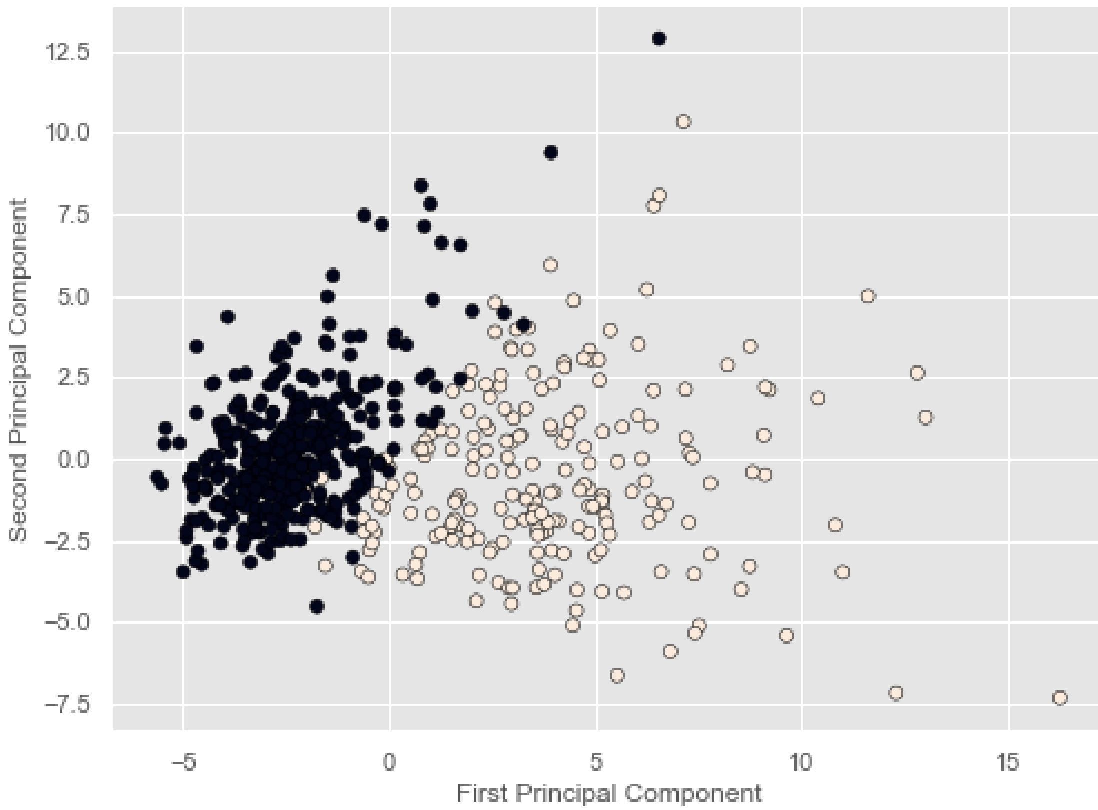
Essentially the same process for each of the above two steps:

i) import
ii) instantiate
iii) fit
iv) transform

[122]:
```
### Applying Dimensionality Reduction

from sklearn.preprocessing import StandardScaler  # Import
scaler = StandardScaler() # Instantiate
scaler.fit(data) # Fit
scaled_data = scaler.transform(data) # Transform

# Applying PCA
from sklearn.decomposition import PCA  # Import

pca = PCA(n_components=2)  # Instantiate
pca.fit(scaled_data)  # Fit
X_pca = pca.transform(scaled_data)  # Transform
print(" Data dimensions before reduction:",scaled_data.shape)
print(" Data dimensions after reduction:",X_pca.shape)


# visualization of PCA
plt.figure(figsize=(8,6))
plt.scatter(X_pca[:,0],X_pca[:,1], c=data['diagnosis'],edgecolor='black')
#plt.scatter(X_pca[:,0],X_pca[:,1], edgecolor='black')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
```

```
 Data dimensions before reduction: (569, 31)
 Data dimensions after reduction: (569, 2)
```

[122]:
```
Text(0, 0.5, 'Second Principal Component')
```



[123]:

```
print(pca.components_)
```

```
[[ 0.21691695  0.21639914  0.10359936  0.2245478   0.21796464  0.13764549
   0.23150388  0.25122179  0.2552664   0.1330126   0.05797189  0.20090409
   0.01547414  0.20563036  0.19798194  0.01123624  0.1619218   0.14578435
   0.1762679   0.03832541  0.09508414  0.22558983  0.10501867  0.23364163
   0.22196962  0.12530106  0.20447639  0.22307483  0.24628844  0.12066328
   0.12672101]
 [-0.07760994 -0.22654473 -0.05826196 -0.20762951 -0.22322434  0.18876038
   0.15847307  0.06786768 -0.02722028  0.19321967  0.36761956 -0.09715581
   0.09118826 -0.08069084 -0.14355563  0.20531469  0.23886735  0.203226
   0.13704859  0.1860032   0.28433173 -0.21289639 -0.04513536 -0.19259202
  -0.21188718  0.17247591  0.14766283  0.10308807 -0.00243309  0.14206245
   0.27644912]]
```

[124]:

```
print(pca.explained_variance_)
```

```
[13.94227406  5.73643378]
```

[125]:

```
#checking normalization with mean and standard deviation
print('Mean after applying PCA',np.mean(X_pca))
```

```
print('Standard Deviation after applying PCA',np.std(X_pca))

from scipy.stats import kurtosis
print('Kurtosis after applying PCA',kurtosis(X_pca))
```

```
Mean after applying PCA -2.4975140097015822e-17
Standard Deviation after applying PCA 3.1340168409862255
Kurtosis after applying PCA [0.6508818 2.857848 ]
```

Data is in normal distribution after applying PCA here. No need to further process.

[126]:
```
# converting the normalized features into a tabular format with the help of␣
↪DataFrame.
pca_Df = pd.DataFrame(data = X_pca
              , columns = ['principal component 1', 'principal component 2'])
pca_Df.head()
```

[126]:

	principal component 1	principal component 2
0	9.225770	2.116196
1	2.655802	-3.784776
2	5.892492	-1.005579
3	7.135401	10.318716
4	4.129423	-1.905579

[127]:
```
pca_Df.corr()
```

[127]:

	principal component 1	principal component 2
principal component 1	1.000000e+00	-8.882348e-17
principal component 2	-8.882348e-17	1.000000e+00

High complexity associated with dataframe having a big number of dimensions/features, which frequently make the target function quite complex and may lead to model overfitting as long as often the dataset lies on the lower dimensionality manifold.

As PCA convert more number of features columns to 2 dimensional data, we have output for 2 columns as principal component 1 and principal component 2.

So that we have reduced set of correlation between attribuutes,

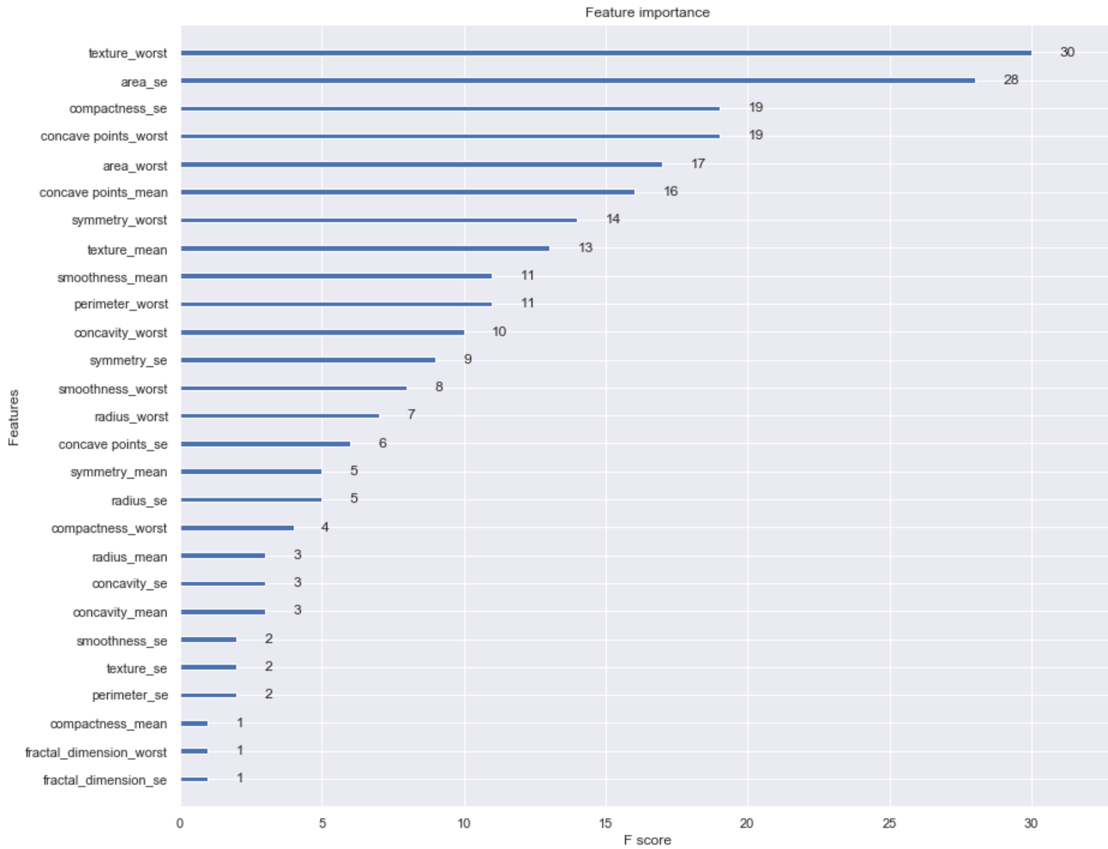
[128]:
```
# Shuffling the Data Set
from sklearn.utils import shuffle
X_pca = shuffle(X_pca)

# Splitting the data set into train and test set
from sklearn.model_selection import train_test_split

features = data.drop(columns = ['diagnosis'])
target = data['diagnosis']
```

```
X_train1, X_test1, y_train1, y_test1 = train_test_split(features, target,␣
 ↪test_size = 0.3,random_state = 0)

print ("Train data set size : ", X_train1.shape)
print ("Test data set size : ", X_test1.shape)
```

```
Train data set size :  (398, 30)
Test data set size :  (171, 30)
```

Feature importance A benefit of using gradient boosting is that after the boosted trees are constructed, it is relatively straightforward to retrieve importance scores for each attribute.

[129]:
```
# Plotting the feature importances using the Boosted Gradient Descent
from xgboost import XGBClassifier
from xgboost import plot_importance

# Training the model
model = XGBClassifier()
model_importance = model.fit(X_train1, y_train1)

# Plotting the Feature importance bar graph
plt.rcParams['figure.figsize'] = [14,12]
sns.set(style = 'darkgrid')
plot_importance(model_importance);
```



2 Preprocessing of Continuous Data

Adult / Census Income dataset Data can be downloaded from following link https://archive.ics.uci.edu/ml/datasets/census+income

[130]:
```
# Import libraries necessary for this project
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import seaborn as sns
sns.set(style="darkgrid")
from time import time

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OneHotEncoder
import scipy.stats as stats
from scipy.stats import kurtosistest

# displaying for notebooks
```

```
%matplotlib inline

columns = ['age', 'workclass', 'fnlwgt', 'education', 'education-num',␣
↪'marital-status', 'occupation',
           'relationship', 'race','sex', 'capital-gain', 'capital-loss',␣
↪'hours-per-week', 'native-country', 'income']

# Load the Census dataset
census = pd.read_csv('dataConti/adult.data', header=None, names=columns,␣
↪skipinitialspace=True)


# Success - Display the first 5 record
display(census.head())
```

```
   age         workclass  fnlwgt  education  education-num  \
0   39         State-gov   77516  Bachelors             13
1   50  Self-emp-not-inc   83311  Bachelors             13
2   38           Private  215646    HS-grad              9
3   53           Private  234721       11th              7
4   28           Private  338409  Bachelors             13

       marital-status         occupation   relationship   race     sex  \
0       Never-married       Adm-clerical  Not-in-family  White    Male
1  Married-civ-spouse    Exec-managerial        Husband  White    Male
2            Divorced  Handlers-cleaners  Not-in-family  White    Male
3  Married-civ-spouse  Handlers-cleaners        Husband  Black    Male
4  Married-civ-spouse     Prof-specialty           Wife  Black  Female

   capital-gain  capital-loss  hours-per-week native-country income
0          2174             0              40  United-States  <=50K
1             0             0              13  United-States  <=50K
2             0             0              40  United-States  <=50K
3             0             0              40  United-States  <=50K
4             0             0              40           Cuba  <=50K
```

[131]:
```
# checking shape of data set
census.shape
```

[131]: (32561, 15)

[132]:
```
import warnings
warnings.filterwarnings("ignore")
```

Data Information age: continuous.

workclass: categorical (Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked)

fnlwgt: continuous.

education: Categoical (Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool)

education-num: continuous.

marital-status: Categorical

relationship: Categorical

race: Categorical

sex: Female, Male - categorical

capital-gain: continuous (income from investment sources, apart from wages/salary)

capital-loss: continuous (losses from investment sources, apart from wages/salary)

hours-per-week: continuous.

native-country: categorical.

[133]:
```
#finding out numerical and categorical

numerical_features=[feature for feature in census.columns if census[feature].
 ↪dtype!='O']
categorical_features=[feature for feature in  census.columns if ␣
 ↪census[feature].dtype=='O' and feature!='wage_class']

print('categorical features: ''\n',categorical_features)
print('\n')
print('numerical features: ''\n',numerical_features)
```

```
categorical features:
 ['workclass', 'education', 'marital-status', 'occupation', 'relationship',
'race', 'sex', 'native-country', 'income']


numerical features:
 ['age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-
week']
```

[134]:
```
# Examine if there are missing value
census.isna().sum()
```

[134]:
```
age               0
workclass         0
fnlwgt            0
education         0
education-num     0
```

```
marital-status    0
occupation        0
relationship      0
race              0
sex               0
capital-gain      0
capital-loss      0
hours-per-week    0
native-country    0
income            0
dtype: int64
```

The result above shows there's no null value in dataset. But according to data notes provided, unknown data was converted into '?'. Therefore, next we'll convert '?' to NaNs and drop unwanted columns.

[135]:
```python
# Drop the fnlwgt column which is useless for later analysis
census = census.drop('fnlwgt', axis=1)

# Read in test data
census_test = pd.read_csv('dataConti/adult.test', header=None, skiprows=1,␣
↪names=columns, skipinitialspace=True)

# Drop the fnlwgt column which is useless for later analysis
census_test = census_test.drop('fnlwgt', axis=1)

# Remove '.' in income column
census_test['income'] = census_test['income'].apply(lambda x: '>50K' if␣
↪x=='>50K.' else '<=50K')

# Review several rows and shape of data set
display(census_test.head())
display(census_test.shape)
```

```
   age  workclass     education  education-num      marital-status  \
0   25    Private          11th              7       Never-married
1   38    Private       HS-grad              9  Married-civ-spouse
2   28  Local-gov    Assoc-acdm             12  Married-civ-spouse
3   44    Private  Some-college             10  Married-civ-spouse
4   18          ?  Some-college             10       Never-married

          occupation relationship   race     sex  capital-gain  capital-loss  \
0  Machine-op-inspct    Own-child  Black    Male             0             0
1    Farming-fishing      Husband  White    Male             0             0
2    Protective-serv      Husband  White    Male             0             0
3  Machine-op-inspct      Husband  Black    Male          7688             0
4                  ?    Own-child  White  Female             0             0
```

```
   hours-per-week native-country income
0              40  United-States  <=50K
1              50  United-States  <=50K
2              40  United-States   >50K
3              40  United-States   >50K
4              30  United-States  <=50K
```

```
(16281, 14)
```

[136]:

```
# Convert '?' to NaNs and remove the entries with NaN value
object_col = census.select_dtypes(include=object).columns.tolist()
for col in object_col:
    census.loc[census[col]=='?', col] = np.nan
    census_test.loc[census_test[col]=='?', col] = np.nan

# Perform an mssing assessment in each column of the dataset.
col_missing_pct = census.isna().sum()/census.shape[0]
col_missing_pct.sort_values(ascending=False)
```

[136]:

```
occupation        0.056601
workclass         0.056386
native-country    0.017905
income            0.000000
hours-per-week    0.000000
capital-loss      0.000000
capital-gain      0.000000
sex               0.000000
race              0.000000
relationship      0.000000
marital-status    0.000000
education-num     0.000000
education         0.000000
age               0.000000
dtype: float64
```

The largest missing percentage by column level is 5% in dataset, and most columns are complete enough. Therefore, here I'll remove the NaN values instead of manually imputing.

[137]:

```
# Removing data entries with missing value
adult_train = census.dropna(axis=0, how='any')
adult_test = census_test.dropna(axis=0, how='any')

# Show the results of the split
print("After removing the missing value:")
print("Training set has {} samples.".format(adult_train.shape[0]))
print("Testing set has {} samples.".format(adult_test.shape[0]))
```

```
After removing the missing value:
Training set has 30162 samples.
Testing set has 15060 samples.
```

[138]:
```
# Finding correlation among features
census.corr() #before removing missing values
```

[138]:
```
                     age  education-num  capital-gain  capital-loss  \
age             1.000000       0.036527      0.077674      0.057775
education-num   0.036527       1.000000      0.122630      0.079923
capital-gain    0.077674       0.122630      1.000000     -0.031615
capital-loss    0.057775       0.079923     -0.031615      1.000000
hours-per-week  0.068756       0.148123      0.078409      0.054256

                hours-per-week
age                   0.068756
education-num         0.148123
capital-gain          0.078409
capital-loss          0.054256
hours-per-week        1.000000
```

We can see correlation changes after removing missing values.

Correlation after removing missing values changes. This means the change in one variable reflects a change in another variable in a predictable pattern then we say that the variables are correlated.

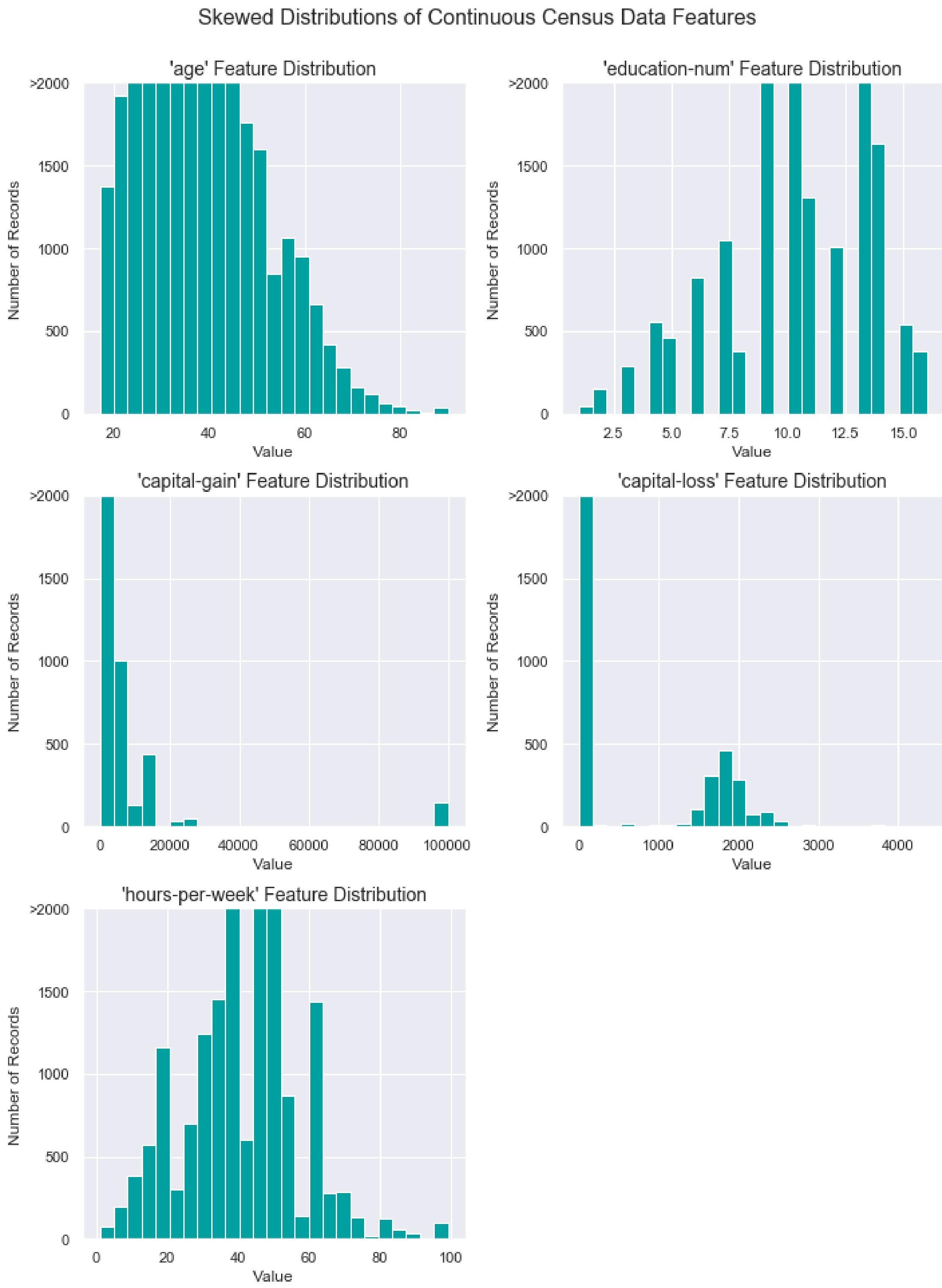
Here it is obeservalble that all are positively correlated except capital-loss and capital-gain as it is negatively correlated.

[139]:
```
adult_train.corr() # after removing missing values
```

[139]:
```
                     age  education-num  capital-gain  capital-loss  \
age             1.000000       0.043526      0.080154      0.060165
education-num   0.043526       1.000000      0.124416      0.079646
capital-gain    0.080154       0.124416      1.000000     -0.032229
capital-loss    0.060165       0.079646     -0.032229      1.000000
hours-per-week  0.101599       0.152522      0.080432      0.052417

                hours-per-week
age                   0.101599
education-num         0.152522
capital-gain          0.080432
capital-loss          0.052417
hours-per-week        1.000000
```

[140]:
```
# Check the skewness of numerical variables in data set
num_col = adult_train.dtypes[adult_train.dtypes != 'object'].index

# Create figure
```

```python
fig = plt.figure(figsize = (10,13));

# Skewed feature plotting
for i, feature in enumerate(adult_train[num_col]):
    ax = fig.add_subplot(3, 2, i+1)
    ax.hist(adult_train[feature], bins = 25, color = '#00A0A0')
    ax.set_title("'%s' Feature Distribution"%(feature), fontsize = 14)
    ax.set_xlabel("Value")
    ax.set_ylabel("Number of Records")
    ax.set_ylim((0, 2000))
    ax.set_yticks([0, 500, 1000, 1500, 2000])
    ax.set_yticklabels([0, 500, 1000, 1500, ">2000"])

# Plot aesthetics
fig.suptitle("Skewed Distributions of Continuous Census Data Features",␣
 ↪fontsize = 16, y = 1.03)

fig.tight_layout()
fig.show()
```



As shown in the graph, there seems skewness in 'capital-gain' and 'capital-loss' features. Use quantitative result to confirm if I need to transform skewness in these two variables.

[141]:
```
# Calculate skew and sort
skew_feats = adult_train[num_col].skew().sort_values(ascending=False)
skewness = pd.DataFrame({'Skew': skew_feats})
skewness
```

[141]:
```
                     Skew
capital-gain    11.902682
capital-loss     4.526380
age              0.530228
hours-per-week   0.330869
education-num   -0.305379
```

Normalizing highly skewed data

[142]:
```
# Split the data into features and target label
income_raw = adult_train['income']
feature_raw = adult_train.drop('income', axis=1)

income_raw_test = adult_test['income']
feature_raw_test = adult_test.drop('income', axis=1)

# Log transform the skewed feature highly-skewed feature 'capital-gain' and␣
↪'capital-loss'.
skewed = ['capital-gain', 'capital-loss']
census_log = pd.DataFrame(data=feature_raw)
census_log[skewed] = feature_raw[skewed].apply(lambda x: np.log(x + 1))

census_log_test = pd.DataFrame(data=feature_raw_test)
census_log_test[skewed] = feature_raw_test[skewed].apply(lambda x: np.log(x +␣
↪1))


# Initialize a scaler, then apply it to the features
scaler = MinMaxScaler() # default=(0, 1)

features_log_minmax_transform = pd.DataFrame(data = census_log)
features_log_minmax_transform[num_col] = scaler.
↪fit_transform(census_log[num_col])

# Transform the test data set
features_log_minmax_transform_test = pd.DataFrame(data = census_log_test)
features_log_minmax_transform_test[num_col] = scaler.
↪transform(census_log_test[num_col])

# Show an example of a record with scaling applied
display(features_log_minmax_transform.head())
display(features_log_minmax_transform_test.head())
```

```
        age         workclass  education  education-num      marital-status  \
0  0.301370         State-gov  Bachelors       0.800000       Never-married
1  0.452055  Self-emp-not-inc  Bachelors       0.800000  Married-civ-spouse
2  0.287671           Private    HS-grad       0.533333            Divorced
3  0.493151           Private       11th       0.400000  Married-civ-spouse
4  0.150685           Private  Bachelors       0.800000  Married-civ-spouse

          occupation   relationship   race     sex  capital-gain  \
0       Adm-clerical  Not-in-family  White    Male      0.667492
1    Exec-managerial        Husband  White    Male      0.000000
2  Handlers-cleaners  Not-in-family  White    Male      0.000000
3  Handlers-cleaners        Husband  Black    Male      0.000000
4     Prof-specialty           Wife  Black  Female      0.000000

   capital-loss  hours-per-week native-country
0           0.0        0.397959  United-States
1           0.0        0.122449  United-States
2           0.0        0.397959  United-States
3           0.0        0.397959  United-States
4           0.0        0.397959           Cuba


        age  workclass     education  education-num      marital-status  \
0  0.109589    Private          11th       0.400000       Never-married
1  0.287671    Private       HS-grad       0.533333  Married-civ-spouse
2  0.150685  Local-gov    Assoc-acdm       0.733333  Married-civ-spouse
3  0.369863    Private  Some-college       0.600000  Married-civ-spouse
5  0.232877    Private          10th       0.333333       Never-married

          occupation   relationship   race   sex  capital-gain  capital-loss  \
0  Machine-op-inspct      Own-child  Black  Male      0.000000           0.0
1    Farming-fishing        Husband  White  Male      0.000000           0.0
2    Protective-serv        Husband  White  Male      0.000000           0.0
3  Machine-op-inspct        Husband  Black  Male      0.777174           0.0
5      Other-service  Not-in-family  White  Male      0.000000           0.0

   hours-per-week native-country
0        0.397959  United-States
1        0.500000  United-States
2        0.397959  United-States
3        0.397959  United-States
5        0.295918  United-States
```

[143]:
```
census_log.corr()
```

[143]:

	capital-gain	capital-loss	hours-per-week
capital-gain	1.000000	-0.067040	0.086243

```
capital-loss       -0.067040      1.000000        0.049468
hours-per-week      0.086243      0.049468        1.000000
```

[144]:
```python
census.corr()
```

[144]:
```
                     age  education-num  capital-gain  capital-loss  \
age             1.000000       0.036527      0.077674      0.057775
education-num   0.036527       1.000000      0.122630      0.079923
capital-gain    0.077674       0.122630      1.000000     -0.031615
capital-loss    0.057775       0.079923     -0.031615      1.000000
hours-per-week  0.068756       0.148123      0.078409      0.054256

                hours-per-week
age                   0.068756
education-num         0.148123
capital-gain          0.078409
capital-loss          0.054256
hours-per-week        1.000000
```

Appling log transformation changes correlation. In this case the two coefficients may lead to different statistical inference. For example, a correlation coefficient of 0.2 is considered to be negligible correlation while a correlation coefficient of 0.3 is considered as low positive correlation.

[145]:
```python
X = feature_raw

y =income_raw
```

[146]:
```python
X.head()
```

[146]:
```
        age          workclass  education  education-num      marital-status  \
0  0.301370          State-gov  Bachelors       0.800000       Never-married
1  0.452055   Self-emp-not-inc  Bachelors       0.800000  Married-civ-spouse
2  0.287671            Private    HS-grad       0.533333            Divorced
3  0.493151            Private       11th       0.400000  Married-civ-spouse
4  0.150685            Private  Bachelors       0.800000  Married-civ-spouse

           occupation   relationship   race     sex  capital-gain  \
0        Adm-clerical  Not-in-family  White    Male      0.667492
1     Exec-managerial        Husband  White    Male      0.000000
2   Handlers-cleaners  Not-in-family  White    Male      0.000000
3   Handlers-cleaners        Husband  Black    Male      0.000000
4      Prof-specialty           Wife  Black  Female      0.000000

   capital-loss  hours-per-week native-country
0           0.0        0.397959  United-States
1           0.0        0.122449  United-States
2           0.0        0.397959  United-States
```

```
3           0.0        0.397959  United-States
4           0.0        0.397959           Cuba
```

[147]:
```python
# splitting data to train and test for further preprocessing

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
↪random_state = 0)
```

[148]:
```python
from sklearn import preprocessing

categorical = ['workclass', 'education', 'marital-status', 'occupation',␣
↪'relationship', 'race', 'sex', 'native-country']
for feature in categorical:
        le = preprocessing.LabelEncoder()
        X_train[feature] = le.fit_transform(X_train[feature])
        X_test[feature] = le.transform(X_test[feature])
```

[149]:
```python
# changing data format to pandas.
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = pd.DataFrame(scaler.fit_transform(X_train), columns = X.columns)

X_test = pd.DataFrame(scaler.transform(X_test), columns = X.columns)
X_train.head()
```

[149]:
```
        age  workclass  education  education-num  marital-status  occupation  \
0  0.417684  -0.204511   0.170452      -0.434574       -1.721729    1.251744
1 -0.494550  -0.204511   1.222721      -0.042490       -0.387728    1.748265
2 -0.646589  -1.261772   0.170452      -0.434574       -1.721729    1.003483
3 -1.178726  -0.204511   1.222721      -0.042490       -0.387728   -0.982600
4  1.177880  -1.261772   1.222721      -0.042490       -0.387728    1.003483

   relationship      race       sex  capital-gain  capital-loss  \
0     -0.266868  0.382579 -1.438780     -0.301995      4.344011
1     -0.890070  0.382579  0.695033     -0.301995     -0.223788
2      0.979538  0.382579 -1.438780     -0.301995     -0.223788
3     -0.890070  0.382579  0.695033     -0.301995     -0.223788
4     -0.890070  0.382579  0.695033     -0.301995     -0.223788

   hours-per-week  native-country
0       -0.078362        0.265946
1        1.605395        0.265946
2       -0.751865        0.265946
```

```
3        1.605395        0.265946
4        0.931892        0.265946
```

[150]:
```python
# applying dimensionality reduction

from sklearn.decomposition import PCA
# Apply PCA to the data
pca = PCA()
model_adult = pca.fit_transform(X_train)
```

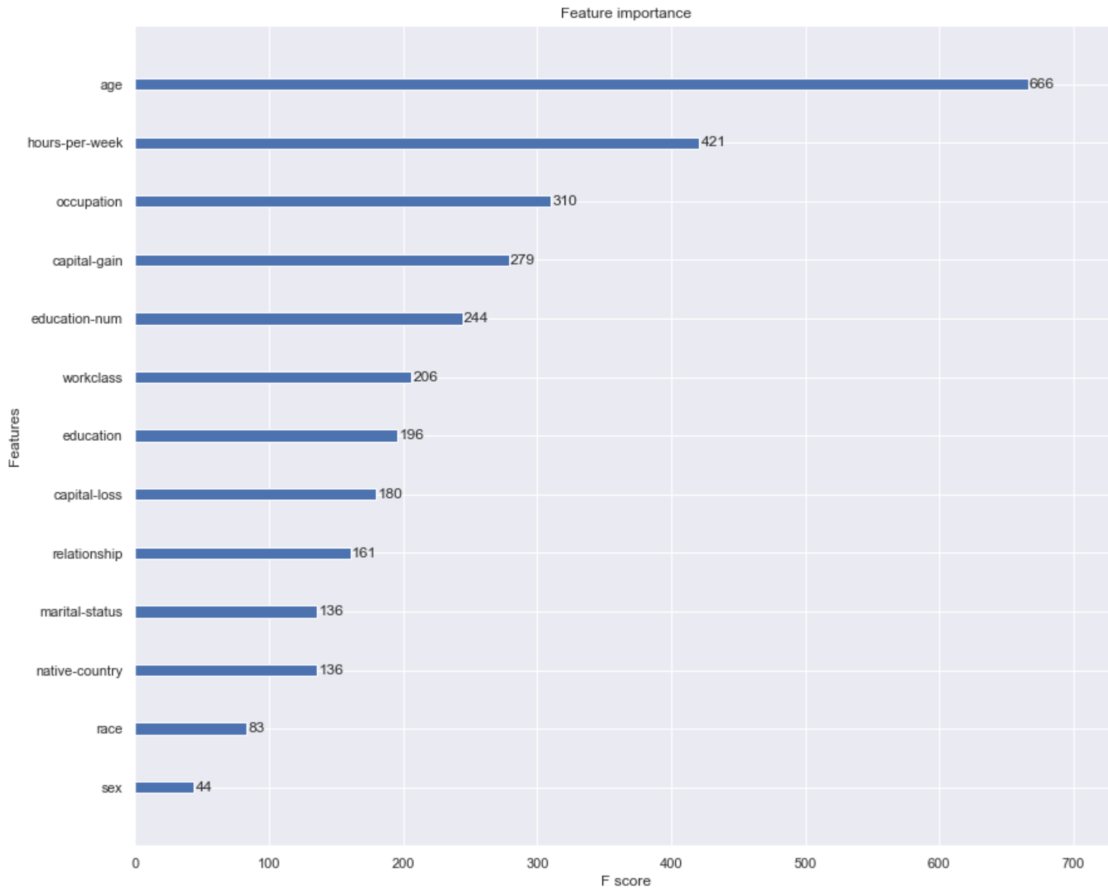
[151]:
```python
vals = pca.explained_variance_ratio_
vals
```

[151]:
```
array([0.16053926, 0.10824834, 0.08848228, 0.08405468, 0.08150518,
       0.07643645, 0.07496046, 0.06805456, 0.06512367, 0.06429608,
       0.05238639, 0.04632909, 0.02958357])
```

[152]:
```python
# Shuffling the Data Set
from sklearn.utils import shuffle
X_adult_pca = shuffle(model_adult)

# Splitting the data set into train and test set
from sklearn.model_selection import train_test_split



print ("Train data set size : ", X_train.shape)
print ("Test data set size : ", X_test.shape)
```

```
Train data set size :  (21113, 13)
Test data set size :  (9049, 13)
```

[153]:
```python
# Plotting the feature importances using the Boosted Gradient Descent
from xgboost import XGBClassifier
from xgboost import plot_importance

# Training the model
model = XGBClassifier()
model_importance_adult = model.fit(X_train, y_train)

# Plotting the Feature importance bar graph
plt.rcParams['figure.figsize'] = [14,12]
sns.set(style = 'darkgrid')
plot_importance(model_importance_adult);
```



3 Transactional Dataset

Data used here is from kaggle can be downloaded from

https://www.kaggle.com/roshansharma/market-basket-optimization

Csv file contains information about Customers buying different grocery items at a Mall.

[154]:
```
#reading data from csv
data3 = pd.read_csv("dataTransactional/store_data.csv", header=None)

print(data3.shape)
```

```
(7501, 20)
```

[155]:
```
# checking first 5 entries
data3.head()
```

[155]:
```
              0         1         2                 3             4  \
0        shrimp   almonds   avocado    vegetables mix  green grapes
```

```
1          burgers   meatballs         eggs                NaN           NaN
2          chutney         NaN          NaN                NaN           NaN
3           turkey     avocado          NaN                NaN           NaN
4    mineral water        milk   energy bar   whole wheat rice     green tea

                  5     6                7              8              9  \
0  whole weat flour  yams   cottage cheese   energy drink   tomato juice
1               NaN   NaN              NaN            NaN            NaN
2               NaN   NaN              NaN            NaN            NaN
3               NaN   NaN              NaN            NaN            NaN
4               NaN   NaN              NaN            NaN            NaN

                 10          11      12      13             14      15  \
0    low fat yogurt   green tea   honey   salad   mineral water  salmon
1               NaN         NaN     NaN     NaN            NaN     NaN
2               NaN         NaN     NaN     NaN            NaN     NaN
3               NaN         NaN     NaN     NaN            NaN     NaN
4               NaN         NaN     NaN     NaN            NaN     NaN

                  16                17        18          19
0  antioxydant juice   frozen smoothie   spinach   olive oil
1                NaN               NaN       NaN         NaN
2                NaN               NaN       NaN         NaN
3                NaN               NaN       NaN         NaN
4                NaN               NaN       NaN         NaN
```

[156]:
```python
# checking dataframe information
data3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7501 entries, 0 to 7500
Data columns (total 20 columns):
0     7501 non-null object
1     5747 non-null object
2     4389 non-null object
3     3345 non-null object
4     2529 non-null object
5     1864 non-null object
6     1369 non-null object
7     981 non-null object
8     654 non-null object
9     395 non-null object
10    256 non-null object
11    154 non-null object
12    87 non-null object
13    47 non-null object
14    25 non-null object
```

```
15    8 non-null object
16    4 non-null object
17    4 non-null object
18    3 non-null object
19    1 non-null object
dtypes: object(20)
memory usage: 1.1+ MB
```

[157]:
```python
#missing values

data3.isna().sum()
```

[157]:
```
0        0
1     1754
2     3112
3     4156
4     4972
5     5637
6     6132
7     6520
8     6847
9     7106
10    7245
11    7347
12    7414
13    7454
14    7476
15    7493
16    7497
17    7497
18    7498
19    7500
dtype: int64
```

[158]:
```python
#creating list of itemsets in basket in different transactions

basket_items = []
for index, row in data3.iterrows():
    cleansed_items = [item for item in row if str(item)!='nan']
    #print(f'basket size: {len(cleansed_items)}, basket:\n{cleansed_items}')
    basket_items.append(cleansed_items)

basket_items[:3]
```

[158]:
```
[['shrimp',
  'almonds',
  'avocado',
```

```
  'vegetables mix',
  'green grapes',
  'whole weat flour',
  'yams',
  'cottage cheese',
  'energy drink',
  'tomato juice',
  'low fat yogurt',
  'green tea',
  'honey',
  'salad',
  'mineral water',
  'salmon',
  'antioxydant juice',
  'frozen smoothie',
  'spinach',
  'olive oil'],
 ['burgers', 'meatballs', 'eggs'],
 ['chutney']]
```

[159]:
```
#Creating transaction DataFrame - this will helps to treate missing values in␣
↪the dataset.
from mlxtend.preprocessing import TransactionEncoder
tran_encod = TransactionEncoder()
tran_encod_list = tran_encod.fit(basket_items).transform(basket_items)
transaction_df = pd.DataFrame(tran_encod_list, columns=tran_encod.columns_)
transaction_df.head()
```

[159]:
```
     asparagus  almonds  antioxydant juice  asparagus  avocado  babies food  \
0        False     True               True      False     True        False
1        False    False              False      False    False        False
2        False    False              False      False    False        False
3        False    False              False      False     True        False
4        False    False              False      False    False        False

   bacon  barbecue sauce  black tea  blueberries  …  turkey  vegetables mix  \
0  False           False      False        False  …   False            True
1  False           False      False        False  …   False           False
2  False           False      False        False  …   False           False
3  False           False      False        False  …    True           False
4  False           False      False        False  …   False           False

   water spray  white wine  whole weat flour  whole wheat pasta  \
0        False       False              True              False
1        False       False             False              False
2        False       False             False              False
3        False       False             False              False
```

```
4        False       False                False                 False

   whole wheat rice   yams  yogurt cake  zucchini
0             False   True        False     False
1             False  False        False     False
2             False  False        False     False
3             False  False        False     False
4              True  False        False     False

[5 rows x 120 columns]
```

Above code have solved the problem of missing values

[160]:
```
#checking for missing values
transaction_df.isna().sum()
```

[160]:
```
 asparagus           0
almonds              0
antioxydant juice    0
asparagus            0
avocado              0
                    ..
whole wheat pasta    0
whole wheat rice     0
yams                 0
yogurt cake          0
zucchini             0
Length: 120, dtype: int64
```

[161]:
```
# creating data frame for item freqency
item_count = {}
for col in transaction_df.columns:
    item_count[col] = transaction_df[col].sum()

item_freq_df = pd.DataFrame(data=list(item_count.values()),␣
↪index=list(item_count.keys()), columns=['frequency']).
↪sort_values(by='frequency', ascending=False)
item_freq_df.shape, item_freq_df.head(10)
```

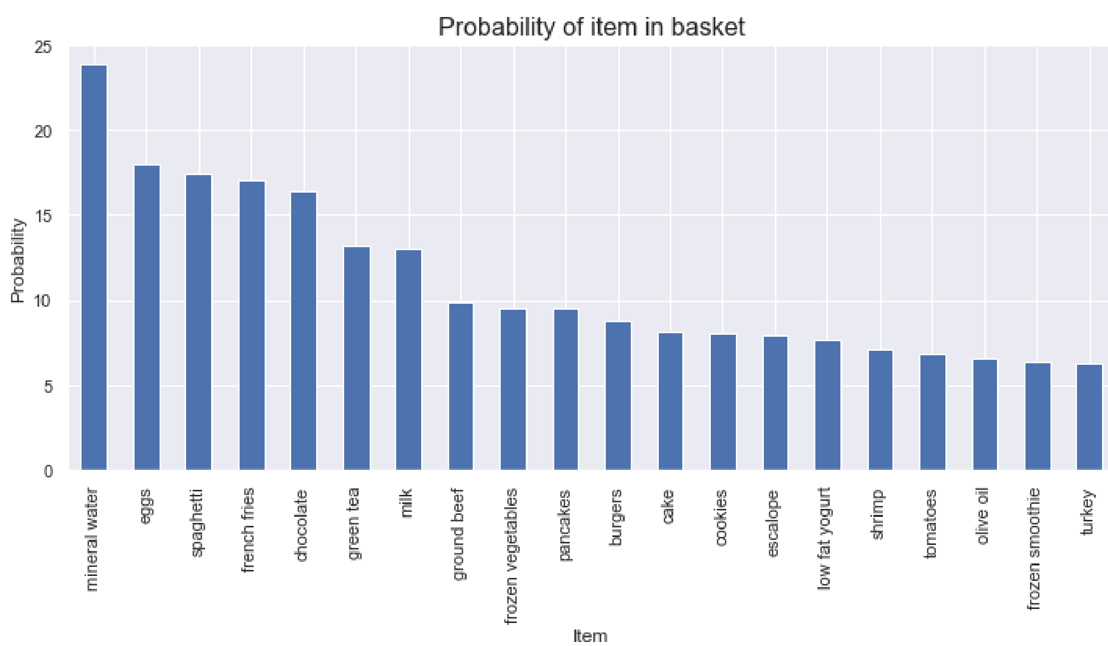
[161]:
```
((120, 1),                      frequency
 mineral water          1788
 eggs                   1348
 spaghetti              1306
 french fries           1282
 chocolate              1229
 green tea               991
 milk                    972
```

```
ground beef              737
frozen vegetables        715
pancakes                 713)
```

Frequency shows the sold item in order of most sold to least sold in the store.

[162]:
```
#Most purchased items
support =transaction_df.mean()
support.sort_values()

(100*support.sort_values(ascending=False))[:20].
 ↪plot(kind='bar',grid=True,figsize = (12,5))
plt.title("Probability of item in basket",fontsize = 16)
plt.xlabel('Item')
plt.ylabel('Probability')
```

[162]:
```
Text(0, 0.5, 'Probability')
```



4 Clustering 1 - Normal

Dataset is available on kaggle.

Titanic Dataset - Prediction of survived.

https://www.kaggle.com/c/titanic/data

Dataset Information: survival binary pclass - (Ticket class 1 = 1st, 2 = 2nd, 3 = 3rd) categorical sex - categorical,nominal

Age - continous,ratio

sibsp # of siblings / spouses aboard the Titanic - nominal

parch # of parents / children aboard the Titanic -nominal

ticket - nominal

fare - ordinal

cabin - nominal

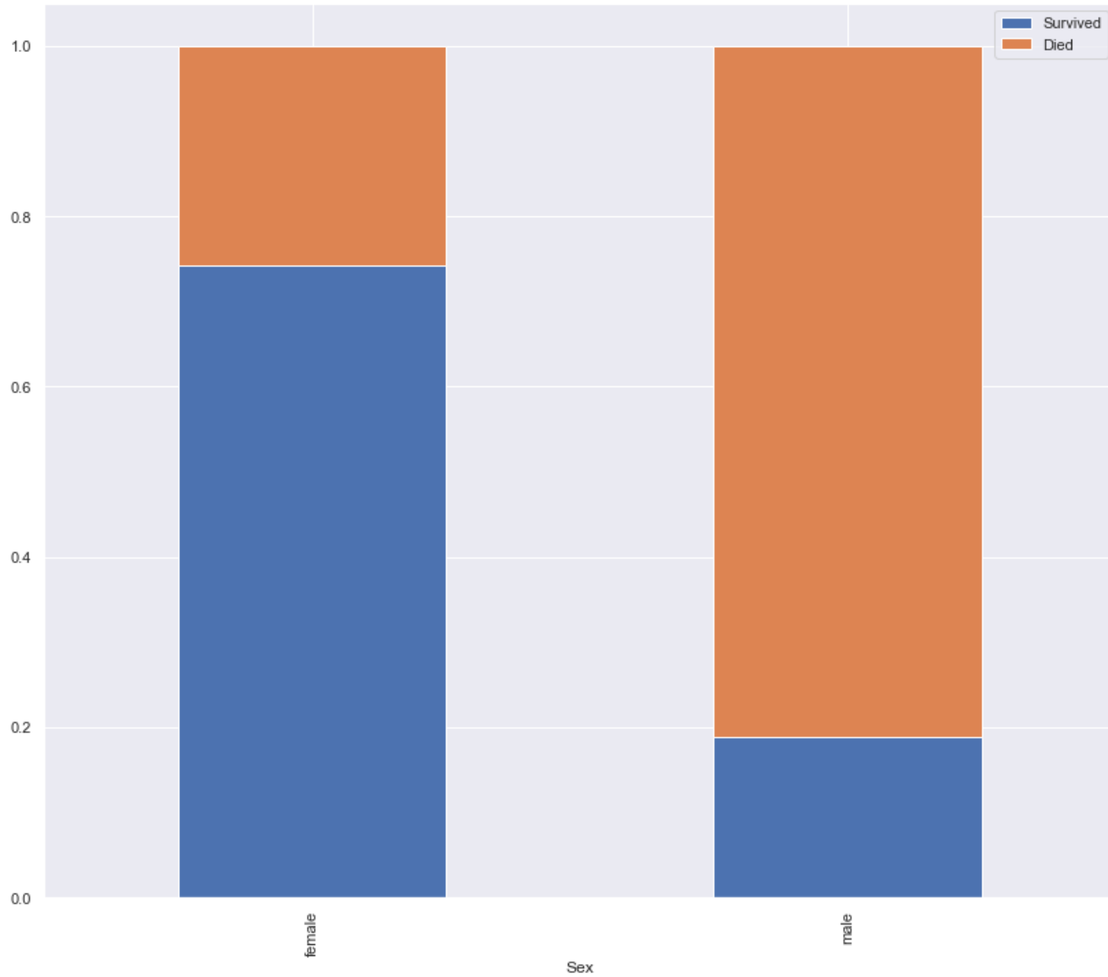
embarked (Port of Embarkation C = Cherbourg, Q = Queenstown, S = Southampton) - nominal

[163]:
```
#importing data

df= pd.read_csv("dataClustering/train.csv")
df_test= pd.read_csv("dataClustering/test.csv")
df.head()
```

[163]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

[164]:
```
#### Visualizing Survival based on gender
```

[165]:
```
df['Died'] = 1 - df['Survived']

df.groupby('Sex').agg('mean')[['Survived', 'Died']].plot(kind='bar',␣
↪stacked=True);
```



Observation- female were more likely to survive more than men

[166]:
```
## Finding out the missing values

df.isna().sum().sort_values(ascending = False)
```

[166]:
```
Cabin       687
Age         177
Embarked      2
Died          0
Fare          0
Ticket        0
Parch         0
SibSp         0
Sex           0
Name          0
Pclass        0
```

```
Survived         0
PassengerId      0
dtype: int64
```

[167]:
```
# is age and cabin highly correlated with the label we are interested in␣
 ↪predicting?
# checking correlation for that
df.corr()
```

[167]:
```
             PassengerId  Survived    Pclass       Age     SibSp     Parch  \
PassengerId     1.000000 -0.005007 -0.035144  0.036847 -0.057527 -0.001652
Survived       -0.005007  1.000000 -0.338481 -0.077221 -0.035322  0.081629
Pclass         -0.035144 -0.338481  1.000000 -0.369226  0.083081  0.018443
Age             0.036847 -0.077221 -0.369226  1.000000 -0.308247 -0.189119
SibSp          -0.057527 -0.035322  0.083081 -0.308247  1.000000  0.414838
Parch          -0.001652  0.081629  0.018443 -0.189119  0.414838  1.000000
Fare            0.012658  0.257307 -0.549500  0.096067  0.159651  0.216225
Died            0.005007 -1.000000  0.338481  0.077221  0.035322 -0.081629

                 Fare      Died
PassengerId  0.012658  0.005007
Survived     0.257307 -1.000000
Pclass      -0.549500  0.338481
Age          0.096067  0.077221
SibSp        0.159651  0.035322
Parch        0.216225 -0.081629
Fare         1.000000 -0.257307
Died        -0.257307  1.000000
```

Age and cabin aren't really "correlated" with our label. And missing values arent making them any more desirable. so I will drop Cabin, Age, Embarked.

[168]:
```
# droping Cabin, Age, Embarked from train data
df.drop({'Cabin', 'Age', 'Embarked'}, axis=1, inplace= True)

# droping also from the test data set as well.

df_test.drop({'Cabin', 'Age', 'Embarked'}, axis=1, inplace= True)
```

[169]:
```
#checking missing values in test
df_test.isna().sum().sort_values(ascending = False)
```

[169]:
```
Fare           1
Ticket         0
Parch          0
SibSp          0
Sex            0
```

```
Name           0
Pclass         0
PassengerId    0
dtype: int64
```

[170]:
```
# Setting up a loop to fill value for that specific row

for i in range(len(df_test['Fare'])):
    if df_test['PassengerId'][i] == 1044:
        df_test['Fare'][i] = 10
```

[171]:
```
# Checking it if the value is filled up or not.

df_test.iloc[[152]]['Fare']
```

[171]:
```
152    10.0
Name: Fare, dtype: float64
```

[172]:
```
# checking updated shape
print(df.shape)
print(df_test.shape)
```

```
(891, 10)
(418, 8)
```

[173]:
```
# checking first five entries of train data
df.head()
```

[173]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex  SibSp  Parch  \
0                            Braund, Mr. Owen Harris    male      1      0
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female      1      0
2                             Heikkinen, Miss. Laina  female      0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female      1      0
4                           Allen, Mr. William Henry    male      0      0

             Ticket     Fare  Died
0         A/5 21171   7.2500     1
1          PC 17599  71.2833     0
2  STON/O2. 3101282   7.9250     0
```

```
3           113803  53.1000     0
4           373450   8.0500     1
```

[174]:
```
# chekking entries of test data
df_test.head()
```

[174]:
```
   PassengerId  Pclass                                          Name     Sex  \
0          892       3                              Kelly, Mr. James    male
1          893       3              Wilkes, Mrs. James (Ellen Needs)  female
2          894       2                     Myles, Mr. Thomas Francis    male
3          895       3                              Wirz, Mr. Albert    male
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)  female

   SibSp  Parch   Ticket     Fare
0      0      0   330911   7.8292
1      1      0   363272   7.0000
2      0      0   240276   9.6875
3      0      0   315154   8.6625
4      1      1  3101298  12.2875
```

PassengerId later needed during predicting survval on testing data, so saving it in another variable.

I need to remove Died from training data and PassengerId from both of the data sets that are less correlated with labels.

[175]:
```
# saving PassengerId
id= df_test['PassengerId']
# Deleting PassengerId and Died
df.drop({'PassengerId', 'Died'}, axis=1, inplace= True)
df_test.drop({'PassengerId'}, axis=1, inplace= True)
```

[176]:
```
#checking updated shape
print(df.shape)
print(df_test.shape)
```

```
(891, 8)
(418, 7)
```

The one extra column in train is the label. so lets take that out:

`train_test_split the train then to get out validation data;`

[177]:
```
df.corr()
```

[177]:
```
          Survived    Pclass     SibSp     Parch      Fare
Survived  1.000000 -0.338481 -0.035322  0.081629  0.257307
Pclass   -0.338481  1.000000  0.083081  0.018443 -0.549500
SibSp    -0.035322  0.083081  1.000000  0.414838  0.159651
Parch     0.081629  0.018443  0.414838  1.000000  0.216225
```

```
Fare      0.257307 -0.549500  0.159651  0.216225  1.000000
```

data dimenisons are reduced so negative correlation is also reduced.

[178]:
```
# Defining labels:
y= df['Survived']
# dropping it from trian:
df.drop({'Survived'}, axis= 1, inplace= True)
```

Since we have categorical variables, we must convert them to some sort of numeric value so that our model could understand and create a relationship between various attributes.

[179]:
```
# converison of categorical to numerical:

df1= df
df2= df_test

df= pd.get_dummies(df)
df_test= pd.get_dummies(df_test)
```

[180]:
```
for col in df.columns:
  if col not in df_test.columns:
    df.drop({col}, axis= 1, inplace= True)

for col in df_test.columns:
  if col not in df.columns:
    df_test.drop({col}, axis= 1, inplace= True)
```

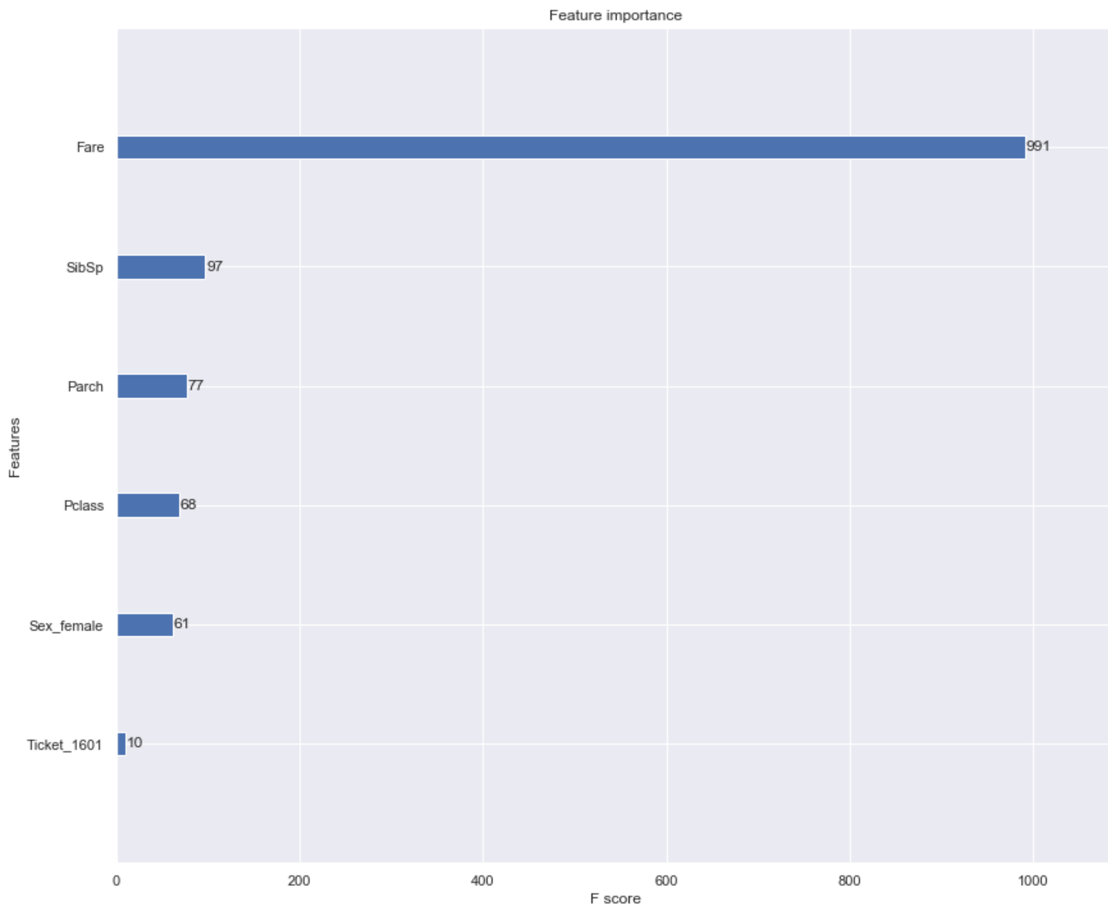
[181]:
```
# Checking out the shapes of both data sets:

print(df.shape)
print(df_test.shape)
```

```
(891, 123)
(418, 123)
```

[182]:
```
from sklearn.model_selection import train_test_split

# Splitting data for training, validation

X_train, X_test, y_train, y_test= train_test_split(df, y, random_state= 42)
```

[183]:
```
# Plotting the feature importances using the Boosted Gradient Descent
from xgboost import XGBClassifier
from xgboost import plot_importance

# Training the model
model = XGBClassifier()
```

```
model_importance = model.fit(X_train, y_train)

# Plotting the Feature importance bar graph
plt.rcParams['figure.figsize'] = [14,12]
sns.set(style = 'darkgrid')
plot_importance(model_importance);
```



Observation :

`One who is paying high fair is more likely to survived.`

`Females are more likely to survive.`

Data is ready for further classification after preprocessing. As I am here in this case focused on survival rate so I had done prprocessing of data in that relative manner. If we want to predict anything else, then we need to perform EDA (Exploratry Data Analysis) to understand how to preprocess.

5 Clustering Text

[184]:
```
import pandas as pd
import numpy as np
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import re
from nltk.corpus import stopwords
import warnings
```

Neural Information Processing Systems (NIPS) is one of the top machine learning conferences in the world. It covers topics ranging from deep learning and computer vision to cognitive science and reinforcement learning. Orignal dataset available- https://github.com/benhamner/nips-papers

The code to scrape and create this dataset is on GitHub.

Dataset used in this assignment by extracted the paper text from the raw PDF files and are releasing that in CSV files.

[185]:
```
#Read datasets/papers.csv into papers
papers = pd.read_csv("dataNips/papers.csv")

# Print out the first rows of papers
papers.head()
```

[185]:
```
     id  year                                              title event_type  \
0     1  1987  Self-Organization of Associative Database and …        NaN
1    10  1987  A Mean Field Theory of Layer IV of Visual Cort…        NaN
2   100  1988  Storing Covariance by the Associative Long-Ter…        NaN
3  1000  1994  Bayesian Query Construction for Neural Network…        NaN
4  1001  1994  Neural Network Ensembles, Cross Validation, an…        NaN

                                            pdf_name          abstract  \
0  1-self-organization-of-associative-database-an…  Abstract Missing
1  10-a-mean-field-theory-of-layer-iv-of-visual-c…  Abstract Missing
2  100-storing-covariance-by-the-associative-long…  Abstract Missing
3  1000-bayesian-query-construction-for-neural-ne…  Abstract Missing
4  1001-neural-network-ensembles-cross-validation…  Abstract Missing

                                          paper_text
0  767\n\nSELF-ORGANIZATION OF ASSOCIATIVE DATABA…
1  683\n\nA MEAN FIELD THEORY OF LAYER IV OF VISU…
2  394\n\nSTORING COVARIANCE BY THE ASSOCIATIVE\n…
3  Bayesian Query Construction for Neural\nNetwor…
4  Neural Network Ensembles, Cross\nValidation, a…
```

[186]:
```
print(papers.paper_text[0][:500] + ' ...')
```

```
767
```

```
SELF-ORGANIZATION OF ASSOCIATIVE DATABASE
AND ITS APPLICATIONS
Hisashi Suzuki and Suguru Arimoto
Osaka University, Toyonaka, Osaka 560, Japan
ABSTRACT
An efficient method of self-organizing associative databases is proposed
together with
applications to robot eyesight systems. The proposed databases can associate any
input
with some output. In the first half part of discussion, an algorithm of self-
organization is
proposed. From an aspect of hardware, it produces a new style of neural netwo
…
```

An example of a text stored in data file

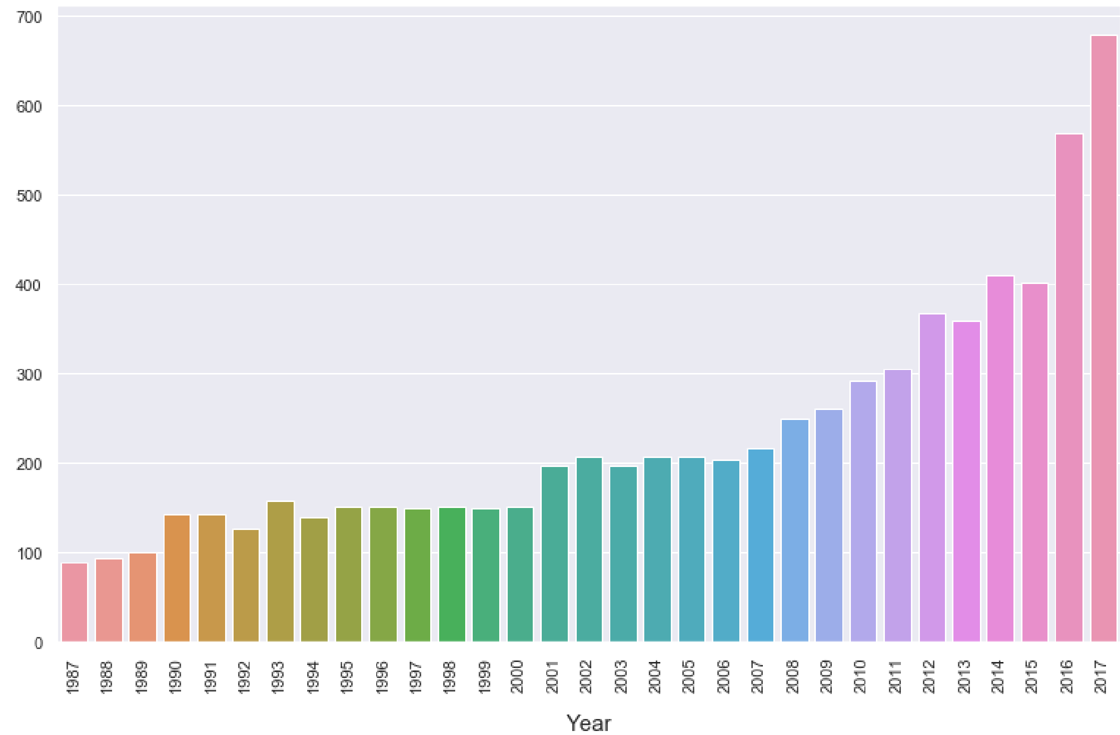
```
[187]: papers = papers.drop(["id", "event_type", "pdf_name"], axis = 1)
papers.head()
```

```
[187]:    year                                              title         abstract  \
0  1987  Self-Organization of Associative Database and …  Abstract Missing
1  1987  A Mean Field Theory of Layer IV of Visual Cort…  Abstract Missing
2  1988  Storing Covariance by the Associative Long-Ter…  Abstract Missing
3  1994  Bayesian Query Construction for Neural Network…  Abstract Missing
4  1994  Neural Network Ensembles, Cross Validation, an…  Abstract Missing

                                          paper_text
0  767\n\nSELF-ORGANIZATION OF ASSOCIATIVE DATABA…
1  683\n\nA MEAN FIELD THEORY OF LAYER IV OF VISU…
2  394\n\nSTORING COVARIANCE BY THE ASSOCIATIVE\n…
3  Bayesian Query Construction for Neural\nNetwor…
4  Neural Network Ensembles, Cross\nValidation, a…
```

For the analysis of the papers, I am only interested in the text data associated with the paper as well as the year the paper was published in.

I will analyse this text data using natural language processing. Since the file contains some meta-data, such as Id and file names, it is necessary to remove all columns that do not contain useful text information.

```
[188]: groups = papers.groupby('year')
counts = groups.size()

plt.figure(figsize = (13, 8))
ax = sns.barplot(counts.index, counts.values)
ax.set_xlabel("Year",fontsize = 15, labelpad = 15)
plt.xticks(rotation = 90)
plt.show()
```



This graph shows number of papers per year.

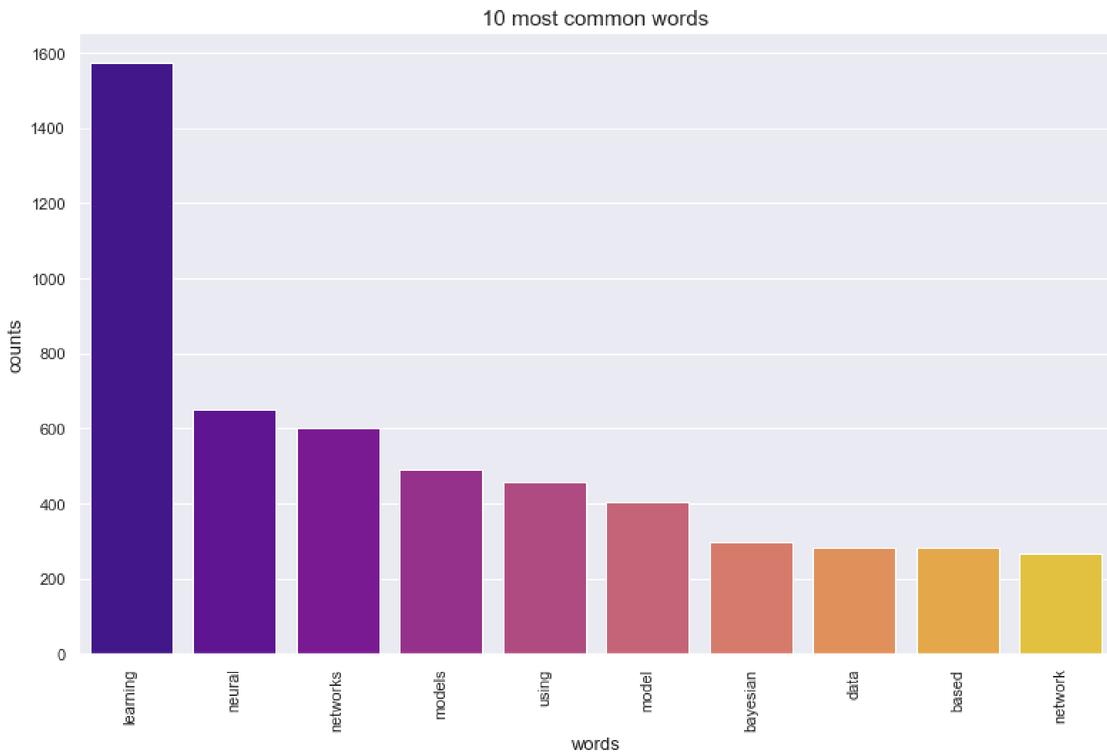
[189]:
```
display(papers['title'].head())

papers['title_processed'] = papers['title'].map(lambda x: re.sub('[,\.!?]', '', x))
papers['title_processed'] = papers['title_processed'].map(str.lower)

display(papers['title_processed'].head())
```

```
0    Self-Organization of Associative Database and ...
1    A Mean Field Theory of Layer IV of Visual Cort...
2    Storing Covariance by the Associative Long-Ter...
3    Bayesian Query Construction for Neural Network...
4    Neural Network Ensembles, Cross Validation, an...
Name: title, dtype: object
```

```
0    self-organization of associative database and ...
1    a mean field theory of layer iv of visual cort...
2    storing covariance by the associative long-ter...
3    bayesian query construction for neural network...
4    neural network ensembles cross validation and ...
Name: title_processed, dtype: object
```

I used a regular expression to remove any punctuation in the title. Then I will perform lowercasing. I'll then print the titles of the first rows before and after applying the modification.

[190]:
```
# plotting 10 most common numbers

from sklearn.feature_extraction.text import CountVectorizer

def plot_10_most_common_words(count_data, count_vectorizer):
    words = count_vectorizer.get_feature_names()
    total_counts = np.zeros(len(words))
    for t in count_data:
        total_counts += t.toarray()[0]

    count_dict = (zip(words, total_counts))
    count_dict = sorted(count_dict, key=lambda x:x[1], reverse=True)
    words = [w[0] for w in count_dict[0:10]]
    counts = [w[1] for w in count_dict[0:10]]
    x_pos = np.arange(len(words))

    sns.barplot(x_pos, counts, palette=("plasma"))
    plt.xticks(x_pos, words, rotation = 90)
    plt.xlabel('words', fontsize = 13)
    plt.ylabel('counts', fontsize = 13)
    plt.title('10 most common words', fontsize = 15)
    plt.show()

    return dict(count_dict)

count_vectorizer = CountVectorizer(stop_words = 'english')
count_data = count_vectorizer.fit_transform(papers['title_processed'])

plt.figure(figsize = (13, 8))
count_dict = plot_10_most_common_words(count_data, count_vectorizer)
```



[191]:
```
#checking null values if exsits.
papers.isnull().sum()
```

[191]:
```
year                0
title               0
abstract            0
paper_text          0
title_processed     0
dtype: int64
```

[192]:
```
# Cleaning unwanted
#Remove numbers,extra chracacters, white spaces if they are not relevant to␣
 ↪your analyses.
#Usually, regular expressions are used to remove numbers.


def cleaned_text(text):
    clean = re.sub("\n"," ",text)
    clean=clean.lower()
    clean=re.sub(r"[~.,%/:;?_&+*=!-]"," ",clean)
    clean=re.sub("[^a-z]"," ",clean)
    clean=clean.lstrip()
    clean=re.sub("\s{2,}"," ",clean)
```

```
    return clean
papers["cleaned_paper_text"]=papers["paper_text"].apply(cleaned_text)
```

[ ]:

[193]:
```
# adding new column to exsisting dataframe as cleaned_paper_text
papers["cleaned_paper_text"] = papers["cleaned_paper_text"].apply(lambda x: ' '.
 ↪join([word for word in x.split() if len(word)>3]))
```

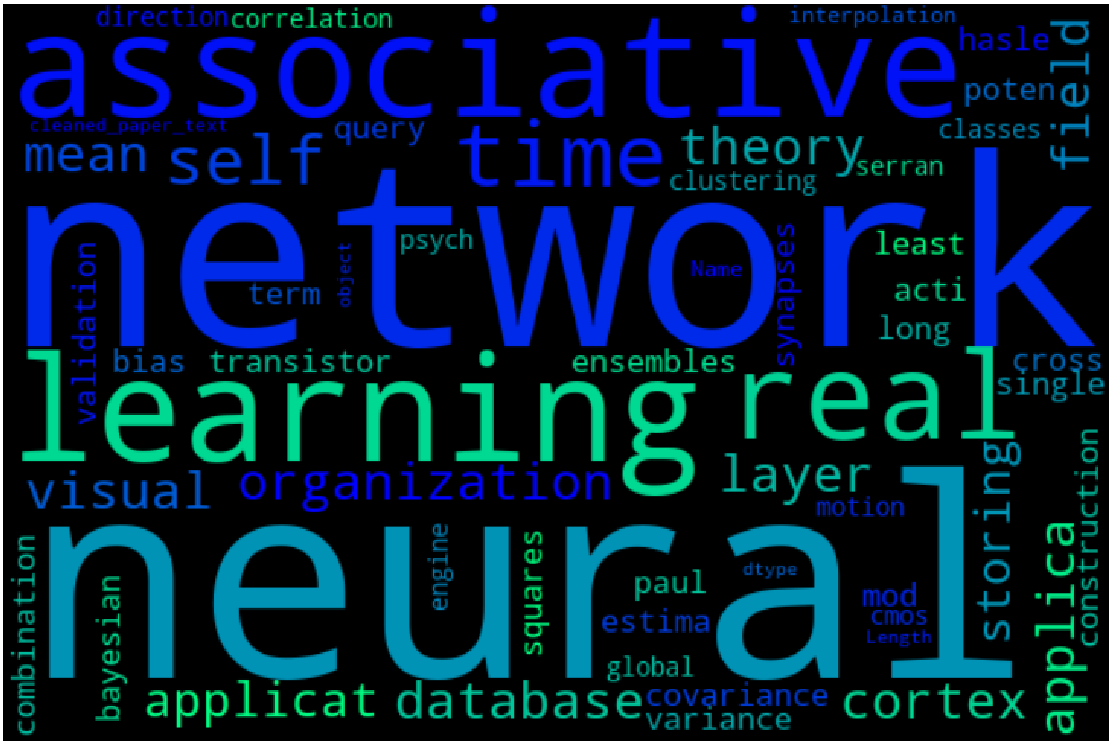
[194]:
```
# display new column
papers["cleaned_paper_text"].head(10)
```

[194]:
```
0    self organization associative database applica…
1    mean field theory layer visual cortex applicat…
2    storing covariance associative long term poten…
3    bayesian query construction neural network mod…
4    neural network ensembles cross validation acti…
5    sing neural instantiate deformable model chris…
6    plasticity mediated competitive learning terre…
7    iceg morphology classification using analogue …
8    real time control tokamak plasma using neural …
9    real time control tokamak plasma using neural …
Name: cleaned_paper_text, dtype: object
```

[ ]:

[195]:
```
# Creating word cloud from cleaned data

from wordcloud import WordCloud
cloud=WordCloud(colormap="winter",width=600,height=400).
 ↪generate(str(papers["cleaned_paper_text"]))
fig=plt.figure(figsize=(13,18))
plt.axis("off")
plt.imshow(cloud,interpolation='bilinear')
```

[195]:
```
<matplotlib.image.AxesImage at 0x2996f329f48>
```



[196]:
```python
# importing english corpus to remove unwanted words usually called as stop words - and, the, if, of, to, etc..
import re, nltk
from nltk.corpus import stopwords

stop=stopwords.words('english')
stop.append("also")
# adding new column to exixting dataframe
papers["stop_removed_paper_text"]=papers["cleaned_paper_text"].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))
```

[197]:
```python
# displaying after removing stop words.
papers["stop_removed_paper_text"].head()
```

[197]:
```
0    self organization associative database applica…
1    mean field theory layer visual cortex applicat…
2    storing covariance associative long term poten…
3    bayesian query construction neural network mod…
4    neural network ensembles cross validation acti…
Name: stop_removed_paper_text, dtype: object
```

Now Checking that all the new generated colummns added in the dataframe or not

[198]:
```
papers.head()
```

[198]:
```
   year                                              title          abstract  \
0  1987  Self-Organization of Associative Database and …  Abstract Missing
1  1987  A Mean Field Theory of Layer IV of Visual Cort…  Abstract Missing
2  1988  Storing Covariance by the Associative Long-Ter…  Abstract Missing
3  1994  Bayesian Query Construction for Neural Network…  Abstract Missing
4  1994  Neural Network Ensembles, Cross Validation, an…  Abstract Missing

                                          paper_text  \
0  767\n\nSELF-ORGANIZATION OF ASSOCIATIVE DATABA…
1  683\n\nA MEAN FIELD THEORY OF LAYER IV OF VISU…
2  394\n\nSTORING COVARIANCE BY THE ASSOCIATIVE\n…
3  Bayesian Query Construction for Neural\nNetwor…
4  Neural Network Ensembles, Cross\nValidation, a…

                                     title_processed  \
0  self-organization of associative database and …
1  a mean field theory of layer iv of visual cort…
2  storing covariance by the associative long-ter…
3  bayesian query construction for neural network…
4  neural network ensembles cross validation and …

                                  cleaned_paper_text  \
0  self organization associative database applica…
1  mean field theory layer visual cortex applicat…
2  storing covariance associative long term poten…
3  bayesian query construction neural network mod…
4  neural network ensembles cross validation acti…

                             stop_removed_paper_text
0  self organization associative database applica…
1  mean field theory layer visual cortex applicat…
2  storing covariance associative long term poten…
3  bayesian query construction neural network mod…
4  neural network ensembles cross validation acti…
```

[199]:
```
papers['stop_removed_paper_text']
```

[199]:
```
0       self organization associative database applica…
1       mean field theory layer visual cortex applicat…
2       storing covariance associative long term poten…
3       bayesian query construction neural network mod…
4       neural network ensembles cross validation acti…
                              …
7236    single transistor learning synapses paul hasle…
7237    bias variance combination least squares estima…
```

```
7238    real time clustering cmos neural engine serran…
7239    learning direction global motion classes psych…
7240    correlation interpolation networks real time e…
Name: stop_removed_paper_text, Length: 7241, dtype: object
```

Tokenization Tokenization is the process of splitting the given text into smaller pieces called tokens. Words, numbers, punctuation marks, and others can be considered as tokens.

[200]:
```
papers["tokenized"]=papers["stop_removed_paper_text"].apply(lambda x: nltk.
↪word_tokenize(x))
```

[201]:
```
papers.head()
```

[201]:
```
   year                                              title         abstract  \
0  1987  Self-Organization of Associative Database and …  Abstract Missing
1  1987  A Mean Field Theory of Layer IV of Visual Cort…  Abstract Missing
2  1988  Storing Covariance by the Associative Long-Ter…  Abstract Missing
3  1994  Bayesian Query Construction for Neural Network…  Abstract Missing
4  1994  Neural Network Ensembles, Cross Validation, an…  Abstract Missing

                                          paper_text  \
0  767\n\nSELF-ORGANIZATION OF ASSOCIATIVE DATABA…
1  683\n\nA MEAN FIELD THEORY OF LAYER IV OF VISU…
2  394\n\nSTORING COVARIANCE BY THE ASSOCIATIVE\n…
3  Bayesian Query Construction for Neural\nNetwor…
4  Neural Network Ensembles, Cross\nValidation, a…

                                     title_processed  \
0  self-organization of associative database and …
1  a mean field theory of layer iv of visual cort…
2  storing covariance by the associative long-ter…
3  bayesian query construction for neural network…
4  neural network ensembles cross validation and …

                                  cleaned_paper_text  \
0  self organization associative database applica…
1  mean field theory layer visual cortex applicat…
2  storing covariance associative long term poten…
3  bayesian query construction neural network mod…
4  neural network ensembles cross validation acti…

                             stop_removed_paper_text  \
0  self organization associative database applica…
1  mean field theory layer visual cortex applicat…
2  storing covariance associative long term poten…
3  bayesian query construction neural network mod…
4  neural network ensembles cross validation acti…
```

```
                                           tokenized
0  [self, organization, associative, database, ap…
1  [mean, field, theory, layer, visual, cortex, a…
2  [storing, covariance, associative, long, term,…
3  [bayesian, query, construction, neural, networ…
4  [neural, network, ensembles, cross, validation…
```

We can see a column added in this dataset as tokenized.

Lematization The aim of lemmatization, like stemming, is to reduce inflectional forms to a common base form. As opposed to stemming, lemmatization does not simply chop off inflections. Instead it uses lexical knowledge bases to get the correct base forms of words.

[202]:
```
# This step take some time ....In my pc it taken around 20 minutes.
import nltk
nltk.download('wordnet')

def word_lemmatizer(text):
    lem_text = [WordNetLemmatizer().lemmatize(i,pos='v') for i in text]
    return lem_text
papers["lemmatized"]=papers["tokenized"].apply(lambda x: word_lemmatizer(x))
papers["lemmatize_joined"]=papers["lemmatized"].apply(lambda x: ' '.join(x))
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\DeLL\AppData\Roaming\nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

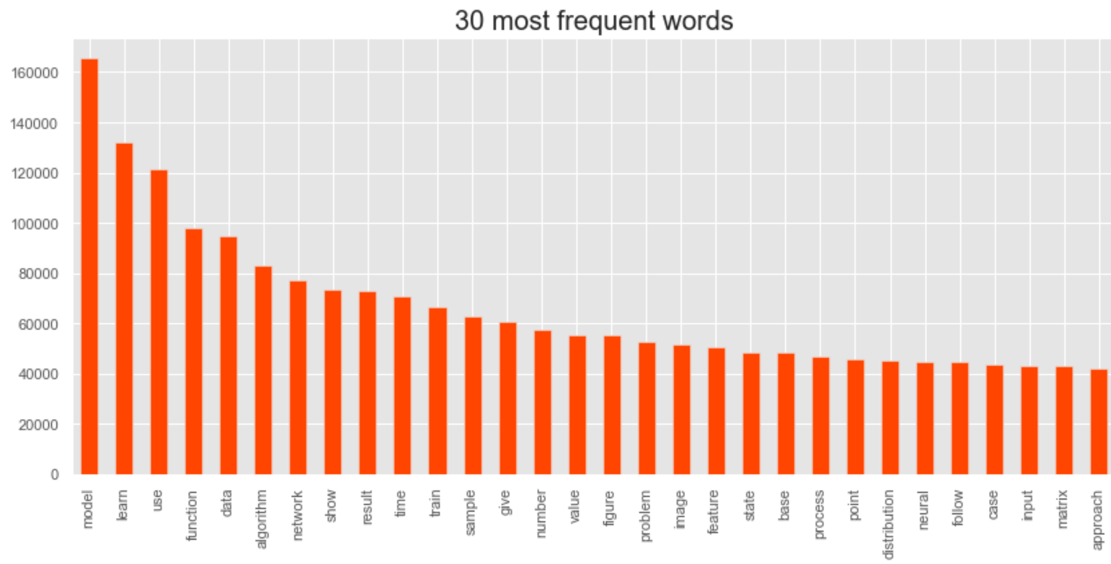
[203]:
```
papers["lemmatize_joined"].head()
```

[203]:
```
0    self organization associative database applica…
1    mean field theory layer visual cortex applicat…
2    store covariance associative long term potenti…
3    bayesian query construction neural network mod…
4    neural network ensembles cross validation acti…
Name: lemmatize_joined, dtype: object
```

[204]:
```
# adding a new column for data analysis
papers['Number_of_words_for_cleaned'] = papers['lemmatize_joined'].apply(lambda␣
 ↪x:len(str(x).split()))
```

[205]:
```
# plotting 30 most frequent words after text preprocessing.
plt.style.use('ggplot')
plt.figure(figsize=(14,6))
freq=pd.Series(" ".join(papers["lemmatize_joined"]).split()).value_counts()[:30]
freq.plot(kind="bar", color = "orangered")
plt.title("30 most frequent words",size=20)
```

```
[205]: Text(0.5, 1.0, '30 most frequent words')
```



This is after preprocessing of text file. As before preprocessing it was diffrent

After the text preprocessing is done, this result may be used for more complicated NLP tasks, for example, machine translation or natural language generation.

```
[ ]:
```

```
[ ]:
```