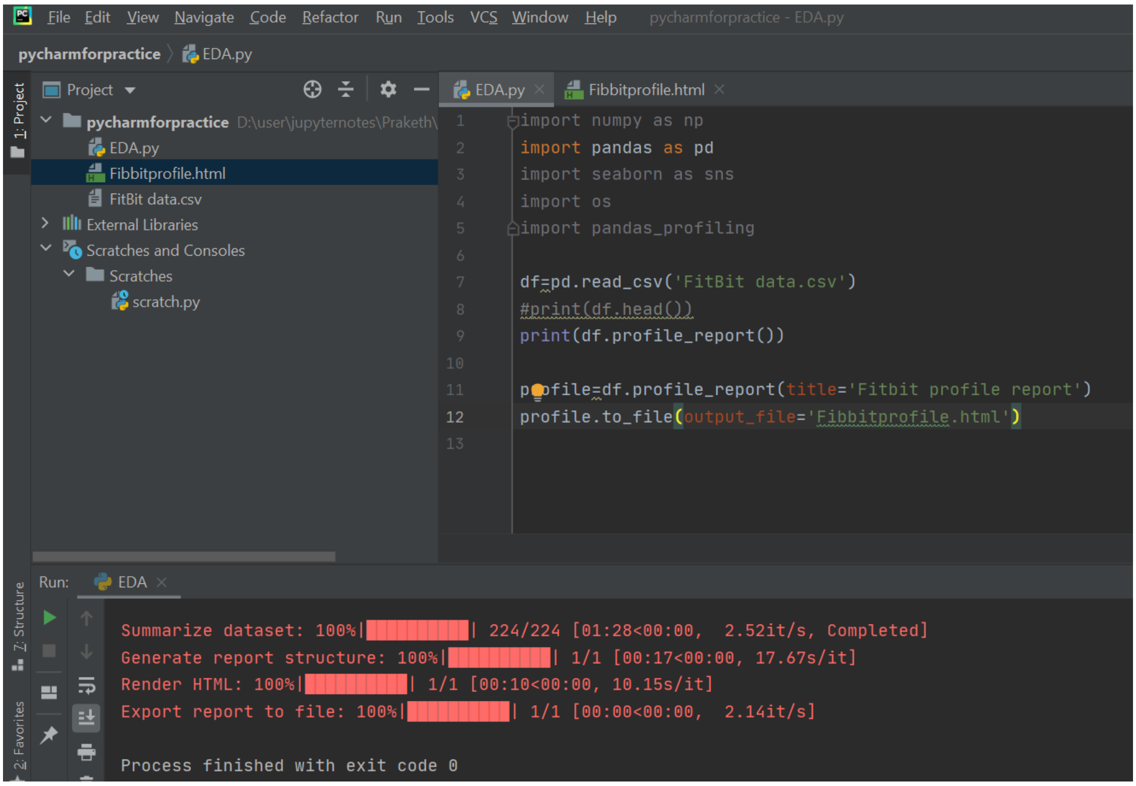
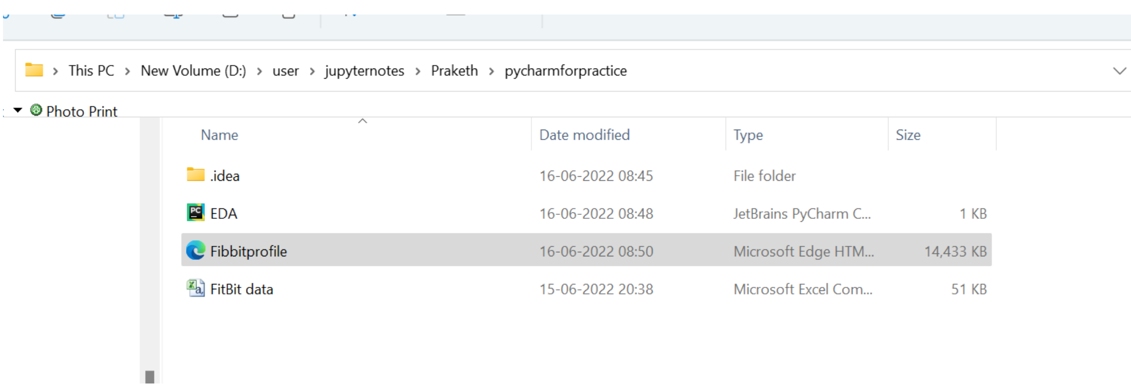
→ Once you read the data set
→ We first try to understand the complete data set
→ What kind of columns given to me what kind of attributes are given to me
→ EDA starts from business understanding,
→ Check on what are numerical data fields what are categorical fields
→ Try to find the relationship between the column, null checks, check on data engineering part of selection part or transformation part, try to understand which column is useful for model building
→ We use pandas_profiling package for getting report on the data for that we need to install
→ Pip install padas-profiling package and we need to import pandas_profiling in our pycharm
→ In pycharm we will not be able to display the profile report on the terminal so we need to store the profile report as .html file as shown below and open and see the .html file in explorer.
→ The html file will contain detailed report of Overview of the data with number of variables, observations, missing cells, duplicate rows total size in memory, Variables report for each of the variable present in your data set, Interactions, Correlations report with graph, sample first and last rows with all columns, missing value graph. This is very useful tool for performing EDA. If you click on the toggle details it gives all the statistics associated with it, mean, median, q1,q2,q3, Range, 5th percentile, 95th percentile, IQR, Histogram, common value details,Extreme value details, 10 miniumum and maximum value details. You can find skewness, kurtosys by looking into the histogram of the data.
→

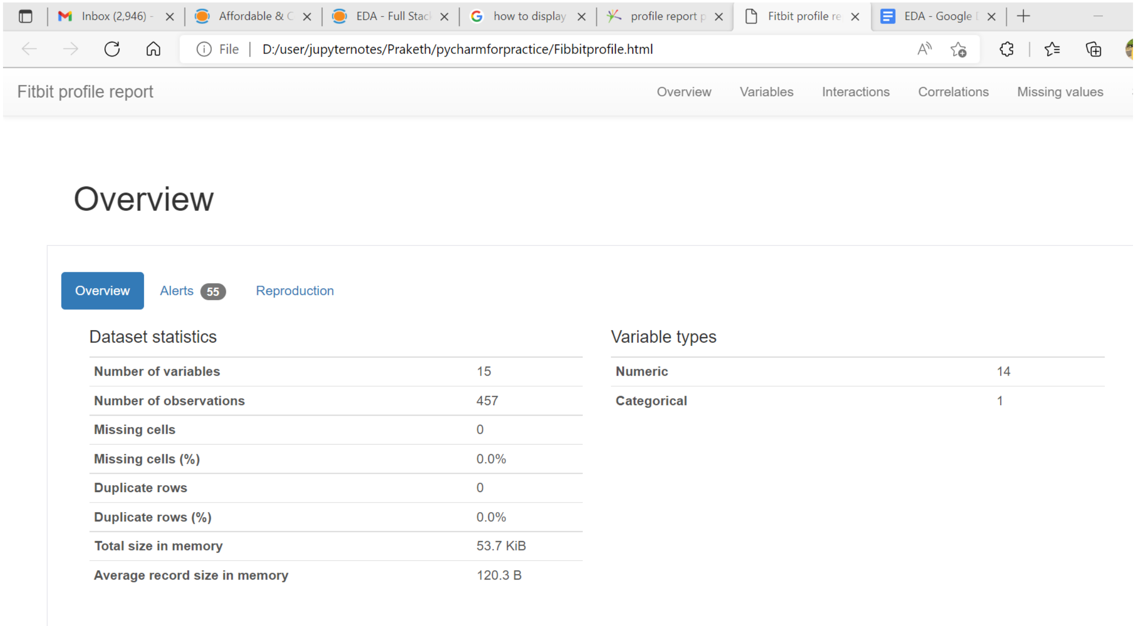


→

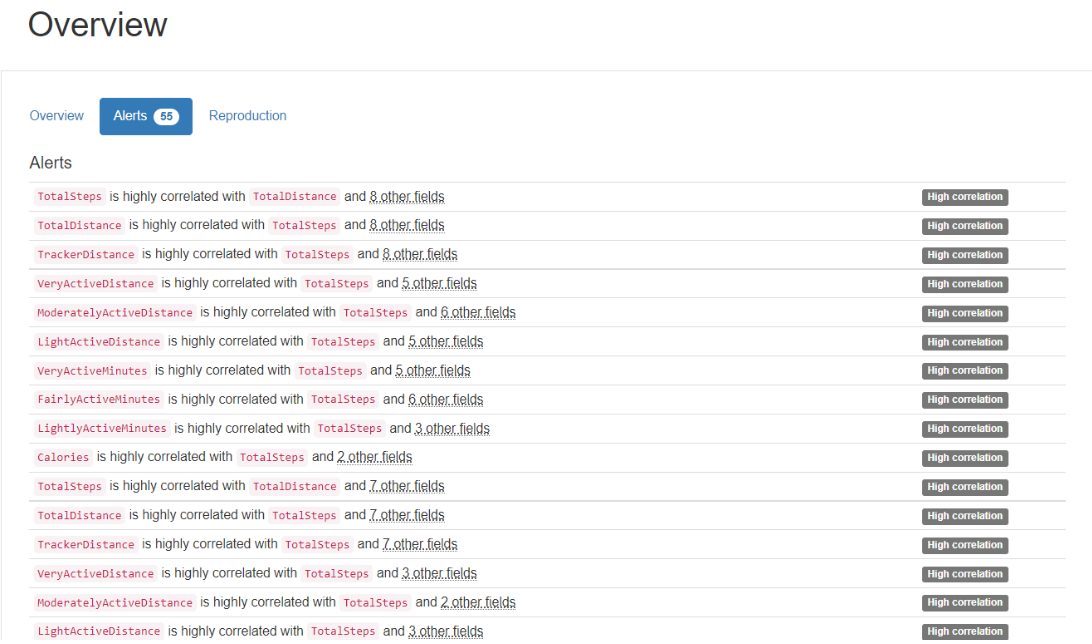


→

- If duplicate rows are there then we need to remove it. We need to check the datatype to know which are categorical data field



-
- If you click on the alters tab in the overview report you get



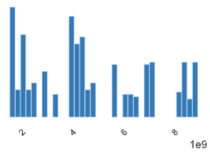
-
- Missing value 0 means no null values. Bar graph is given

Variables

Id
Real number ($\mathbb{R}_{\geq 0}$)
HIGH CORRELATION

Distinct	35
Distinct (%)	7.7%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	4628594643

Minimum	1503960366
Maximum	8877689391
Zeros	0
Zeros (%)	0.0%
Negative	0
Negative (%)	0.0%
Memory size	3.7 KiB



Toggle details

Statistics | Histogram | Common values | Extreme values

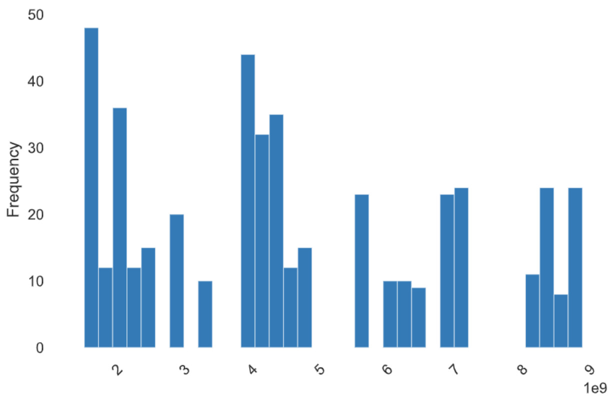
Quantile statistics

Minimum	1503960366
5-th percentile	1624580081
Q1	2347167796
median	4057192912
Q3	6391747486
95-th percentile	8792009665
Maximum	8877689391
Range	7373729025
Interquartile range (IQR)	4044579690

Descriptive statistics

Standard deviation	2293781430
Coefficient of variation (CV)	0.4955675764
Kurtosis	-1.039194515
Mean	4628594643
Median Absolute Deviation (MAD)	2030840877
Skewness	0.3527246238
Sum	$2.115267752 \times 10^{12}$
Variance	$5.261433247 \times 10^{18}$
Monotonicity	Increasing

Statistics | Histogram | Common values | Extreme values



Histogram with fixed size bins (bins=35)

Statistics | Histogram | Common values | Extreme values

Value	Count	Frequency (%)
4057192912	32	7.0%
4020332650	32	7.0%
1503960366	19	4.2%
1624580081	19	4.2%
2347167796	15	3.3%
4702921684	15	3.3%
4445114986	15	3.3%
6962181067	14	3.1%
2320127002	12	2.6%
4558609924	12	2.6%
Other values (25)	272	59.5%

Statistics | Histogram | Common values | Extreme values

Minimum 10 values | Maximum 10 values

Value	Count	Frequency (%)
1503960366	19	4.2%
1624580081	19	4.2%
1644430081	10	2.2%
1844505072	12	2.6%
1927972279	12	2.6%
2022484408	12	2.6%
2026352035	12	2.6%
2320127002	12	2.6%
2347167796	15	3.3%
2873212765	12	2.6%

➔

➔

➔ System has considered date field as categorical data set.

ActivityDate
Categorical

HIGH CORRELATION

Distinct	32
Distinct (%)	7.0%
Missing	0
Missing (%)	0.0%
Memory size	3.7 KiB

4/2/2016	35
4/3/2016	35
4/4/2016	35
4/5/2016	35
4/1/2016	34
Other values (27)	283

Toggle details

➔

Memory size	3.7 KiB

Toggle details

Overview | Categories | Words | Characters

Length

Max length	9
Median length	8
Mean length	8.332603939
Min length	8

Characters and Unicode

Total characters	3808
Distinct characters	11
Distinct categories	2 ?
Distinct scripts	1 ?
Distinct blocks	1 ?

The Unicode Standard assigns character properties to each code point, which can be used to analyse textual variables.

Unique

Unique	0 ?
Unique (%)	0.0%

Sample

1st row	3/25/2016
2nd row	3/26/2016
3rd row	3/27/2016
4th row	3/28/2016
5th row	3/29/2016

➔

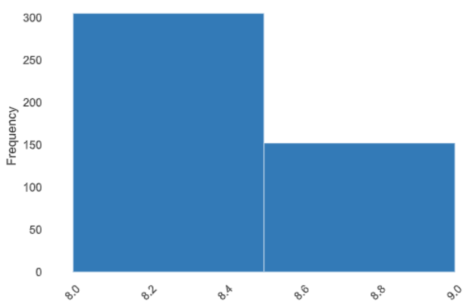
Toggle details

Overview | Categories | Words | Characters

Common Values

Value	Count	Frequency (%)
4/2/2016	35	7.7%
4/3/2016	35	7.7%
4/4/2016	35	7.7%
4/5/2016	35	7.7%
4/1/2016	34	7.4%
4/6/2016	33	7.2%
4/7/2016	33	7.2%
4/8/2016	33	7.2%
4/9/2016	32	7.0%
4/10/2016	29	6.3%
Other values (22)	123	26.9%

Length



Histogram of lengths of the category

➔

Overview | Categories | Words | Characters

Value	Count	Frequency (%)
4/2/2016	35	7.7%
4/5/2016	35	7.7%
4/3/2016	35	7.7%
4/4/2016	35	7.7%
4/1/2016	34	7.4%
4/6/2016	33	7.2%
4/7/2016	33	7.2%
4/8/2016	33	7.2%
4/9/2016	32	7.0%
4/10/2016	29	6.3%
Other values (22)	123	26.9%

➔

Overview | Categories | Words | Characters

Characters | Categories | Scripts | Blocks

Most occurring characters

Value	Count	Frequency (%)
/	914	24.0%
1	623	16.4%
2	556	14.6%
0	499	13.1%
6	496	13.0%
4	422	11.1%
3	135	3.5%
9	44	1.2%
5	41	1.1%
7	39	1.0%

➔

Overview | Categories | Words | Characters

Characters | Categories | Scripts | Blocks

Most occurring categories

Value	Count	Frequency (%)
Decimal Number	2894	76.0%
Other Punctuation	914	24.0%

Most frequent character per category

Decimal Number

Value	Count	Frequency (%)
1	623	21.5%
2	556	19.2%
0	499	17.2%
6	496	17.1%
4	422	14.6%
3	135	4.7%
9	44	1.5%
5	41	1.4%
7	39	1.3%
8	39	1.3%

Other Punctuation

Value	Count	Frequency (%)
/	914	100.0%

➔

Overview Categories Words Characters

Characters Categories Scripts Blocks

Most occurring scripts

Value	Count	Frequency (%)
Common	3808	100.0%

Most frequent character per script

Common

Value	Count	Frequency (%)
/	914	24.0%
1	623	16.4%
2	556	14.6%
0	499	13.1%
6	496	13.0%
4	422	11.1%
3	135	3.5%
9	44	1.2%
5	41	1.1%
7	39	1.0%

➔

Overview Categories Words Characters

Characters Categories Scripts Blocks

Most occurring blocks

Value	Count	Frequency (%)
ASCII	3808	100.0%

Most frequent character per block

ASCII

Value	Count	Frequency (%)
/	914	24.0%
1	623	16.4%
2	556	14.6%
0	499	13.1%
6	496	13.0%
4	422	11.1%
3	135	3.5%
9	44	1.2%
5	41	1.1%
7	39	1.0%

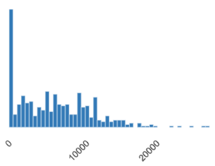
➔

➔

TotalSteps
Real number ($\mathbb{R}_{\geq 0}$)

HIGH CORRELATION
HIGH CORRELATION
HIGH CORRELATION
HIGH CORRELATION
ZEROS

Distinct	389	**Minimum**	0
Distinct (%)	85.1%	**Maximum**	28497
Missing	0	**Zeros**	61
Missing (%)	0.0%	**Zeros (%)**	13.3%
Infinite	0	**Negative**	0
Infinite (%)	0.0%	**Negative (%)**	0.0%
Mean	6546.562363	**Memory size**	3.7 KiB

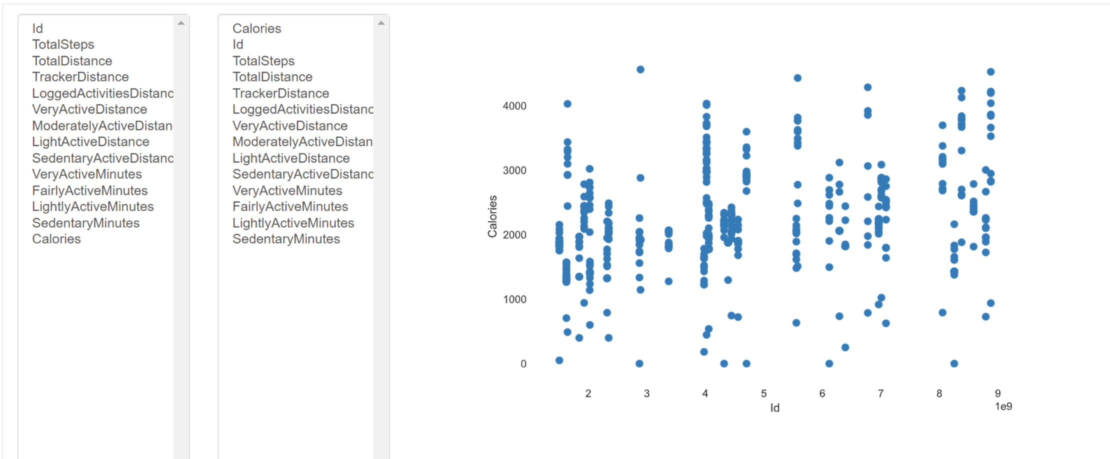


Toggle details

➔

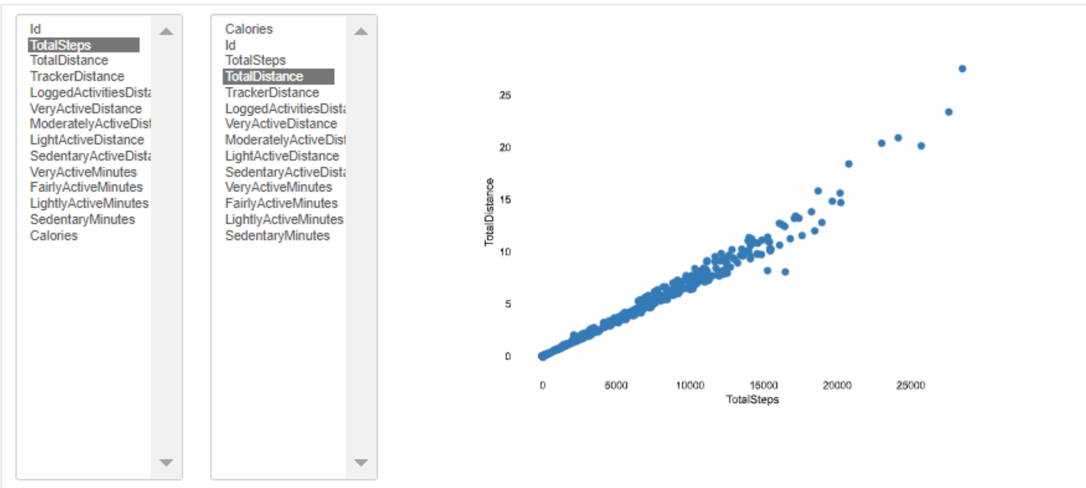
➔ And so on….It gives report like above each variable in the data set

Interactions



➔

Interactions

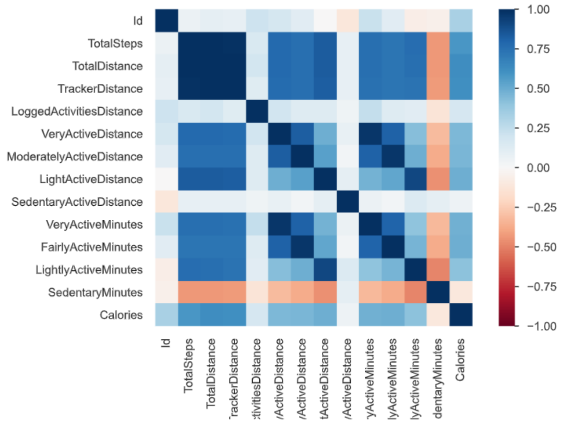


➔

Correlations

Spearman's ρ | Pearson's r | Kendall's τ | Phik (φk)

Toggle correlation descriptions

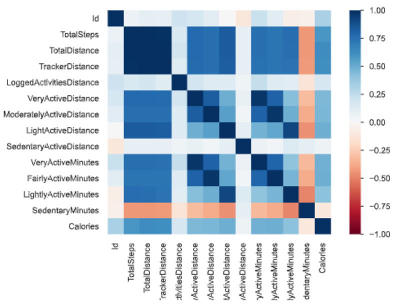


➔

Correlations

Spearman's ρ | Pearson's r | Kendall's τ | Phik (φk)

Toggle correlation descriptions



Spearman's ρ

The Spearman's rank correlation coefficient (*ρ*) is a measure of monotonic correlation between two variables, and is therefore better in catching nonlinear monotonic correlations than Pearson's *r*. It's value lies between -1 and +1, -1 indicating total negative monotonic correlation, 0 indicating no monotonic correlation and 1 indicating total positive monotonic correlation.

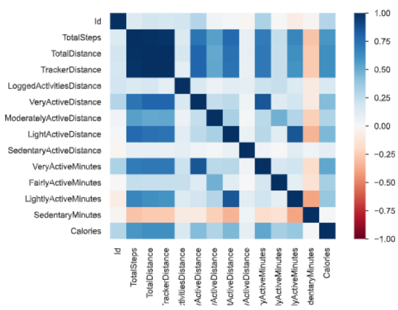
To calculate *ρ* for two variables *X* and *Y*, one divides the covariance of the rank variables of *X* and *Y* by the product of their standard deviations.

➔

Correlations

Spearman's ρ | Pearson's r | Kendall's τ | Phik (φk)

Toggle correlation descriptions



Pearson's r

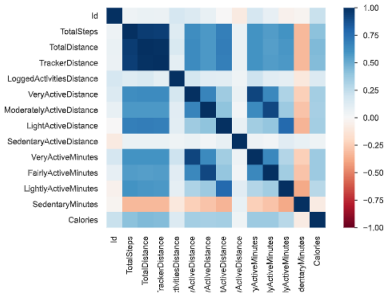
The Pearson's correlation coefficient (*r*) is a measure of linear correlation between two variables. It's value lies between -1 and +1, -1 indicating total negative linear correlation, 0 indicating no linear correlation and 1 indicating total positive linear correlation. Furthermore, *r* is invariant under separate changes in location and scale of the two variables, implying that for a linear function the angle to the x-axis does not affect *r*.

To calculate *r* for two variables *X* and *Y*, one divides the covariance of *X* and *Y* by the product of their standard deviations.

➔

Correlations

Spearman's ρ   Pearson's r   Kendall's τ   Phik (φk)   Toggle correlation descriptions



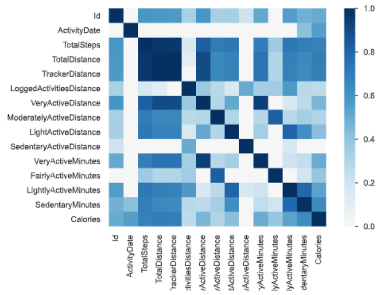
Kendall's τ

Similarly to Spearman's rank correlation coefficient, the Kendall rank correlation coefficient (*τ*) measures ordinal association between two variables. It's value lies between -1 and +1, -1 indicating total negative correlation, 0 indicating no correlation and 1 indicating total positive correlation.

To calculate *τ* for two variables *X* and *Y*, one determines the number of concordant and discordant pairs of observations. *τ* is given by the number of concordant pairs minus the discordant pairs divided by the total number of pairs.

Correlations

Spearman's ρ   Pearson's r   Kendall's τ   Phik (φk)   Toggle correlation descriptions

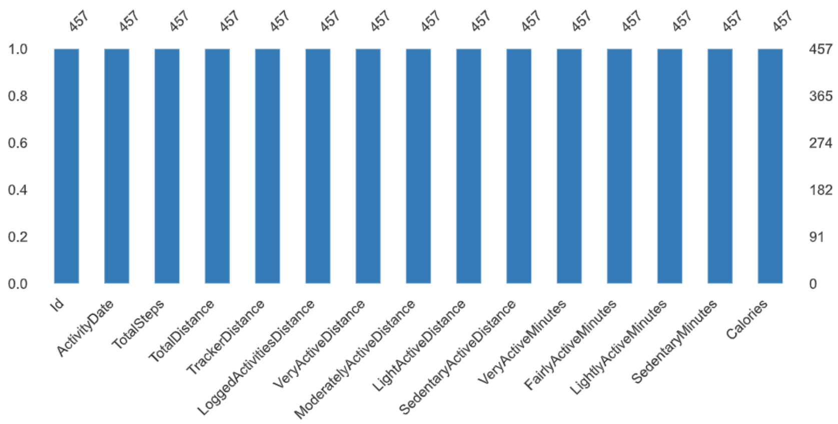


Phik (φk)

Phik (φk) is a new and practical correlation coefficient that works consistently between categorical, ordinal and interval variables, captures non-linear dependency and reverts to the Pearson correlation coefficient in case of a bivariate normal input distribution. There is extensive documentation available here.

Missing values

Count   Matrix



A simple visualization of nullity by column.

Sample

First rows

	Id	ActivityDate	TotalSteps	TotalDistance	TrackerDistance	LoggedActivitiesDistance	VeryActiveDistance	ModeratelyActiveDistance	LightAc
0	1503960366	3/25/2016	11004	7.11	7.11	0.0	2.57	0.46	4.07
1	1503960366	3/26/2016	17609	11.55	11.55	0.0	6.92	0.73	3.91
2	1503960366	3/27/2016	12736	8.53	8.53	0.0	4.66	0.16	3.71
3	1503960366	3/28/2016	13231	8.93	8.93	0.0	3.19	0.79	4.95
4	1503960366	3/29/2016	12041	7.85	7.85	0.0	2.16	1.09	4.61
5	1503960366	3/30/2016	10970	7.16	7.16	0.0	2.36	0.51	4.29
6	1503960366	3/31/2016	12256	7.86	7.86	0.0	2.29	0.49	5.04
7	1503960366	4/1/2016	12262	7.87	7.87	0.0	3.32	0.83	3.64
8	1503960366	4/2/2016	11248	7.25	7.25	0.0	3.00	0.45	3.74
9	1503960366	4/3/2016	10016	6.37	6.37	0.0	0.91	1.28	4.18

➔

Last rows

	Id	ActivityDate	TotalSteps	TotalDistance	TrackerDistance	LoggedActivitiesDistance	VeryActiveDistance	ModeratelyActiveDistance	Light
447	8877689391	4/3/2016	15260	8.190000	8.190000	0.0	1.80	0.75	5.57
448	8877689391	4/4/2016	20779	18.410000	18.410000	0.0	11.73	0.65	6.00
449	8877689391	4/5/2016	10695	8.120000	8.120000	0.0	0.77	0.18	7.09
450	8877689391	4/6/2016	24136	20.910000	20.910000	0.0	12.22	0.54	8.08
451	8877689391	4/7/2016	10910	8.420000	8.420000	0.0	2.96	0.39	5.03
452	8877689391	4/8/2016	23014	20.389999	20.389999	0.0	11.10	0.63	8.62
453	8877689391	4/9/2016	16470	8.070000	8.070000	0.0	0.00	0.02	8.02
454	8877689391	4/10/2016	28497	27.530001	27.530001	0.0	21.92	1.12	4.46
455	8877689391	4/11/2016	10622	8.060000	8.060000	0.0	1.47	0.15	6.37
456	8877689391	4/12/2016	2350	1.780000	1.780000	0.0	0.00	0.00	1.78

Report generated with pandas-profiling.

➔

➔ People ask what kind of EDA I am supposed to do?Answer is whatever you could see in the pandas-profiling report all those type of things you need to perform in your EDA step by step or one by one in your EDA.

➔ df.shape - To get the number of rows and column details

➔ df.head() to get the 5 sample records

➔ df.tail() to ge the last 5 sample records

➔ `print(df.shape)`

➔ `print(df.isnull().sum())  -` To check number of nulls in each column

➔ `print(df.columns)`To get the list of column names

➔ `print(df.dtypes)`To get the data types of column

➔ `print(df["ActivityDate"].unique())`To get the unique values of given column names

➔ Splitting the date columns in to day month year as new columns

➔

➔ `df['newdate']=pd.DatetimeIndex(df['ActivityDate'])` creating a new column

➔ `print(df.dtypes)`

➔ `print(df['newdate'].head())`

➔ `df['year']=pd.DatetimeIndex(df['ActivityDate']).year` creating a new column for year

➔ `df['month']=pd.DatetimeIndex(df['ActivityDate']).month` creating a new column for month

➔ `df['day']=pd.DatetimeIndex(df['ActivityDate']).day` creating a new column for day

→ `df['dayname']=pd.DatetimeIndex(df['ActivityDate']).day_name()` creating a new column for dayname

→ `df['monthname']=pd.DatetimeIndex(df['ActivityDate']).month_name()` creating a new column for month name

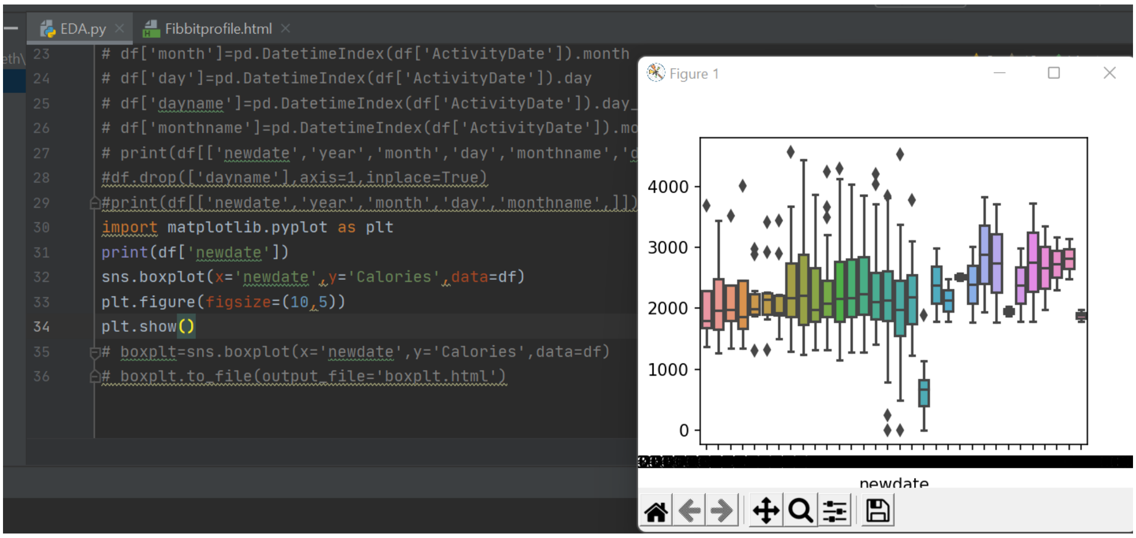
→ `print(df[['newdate','year','month','day','monthname','dayname']])`

	newdate	year	month	day	monthname	dayname
0	2016-03-25	2016	3	25	March	Friday
1	2016-03-26	2016	3	26	March	Saturday
2	2016-03-27	2016	3	27	March	Sunday
3	2016-03-28	2016	3	28	March	Monday

→

→ `df.drop(['dayname'],axis=1,inplace=True) #To drop a column`

→ In pycharm the plots will be shown in popup window



→

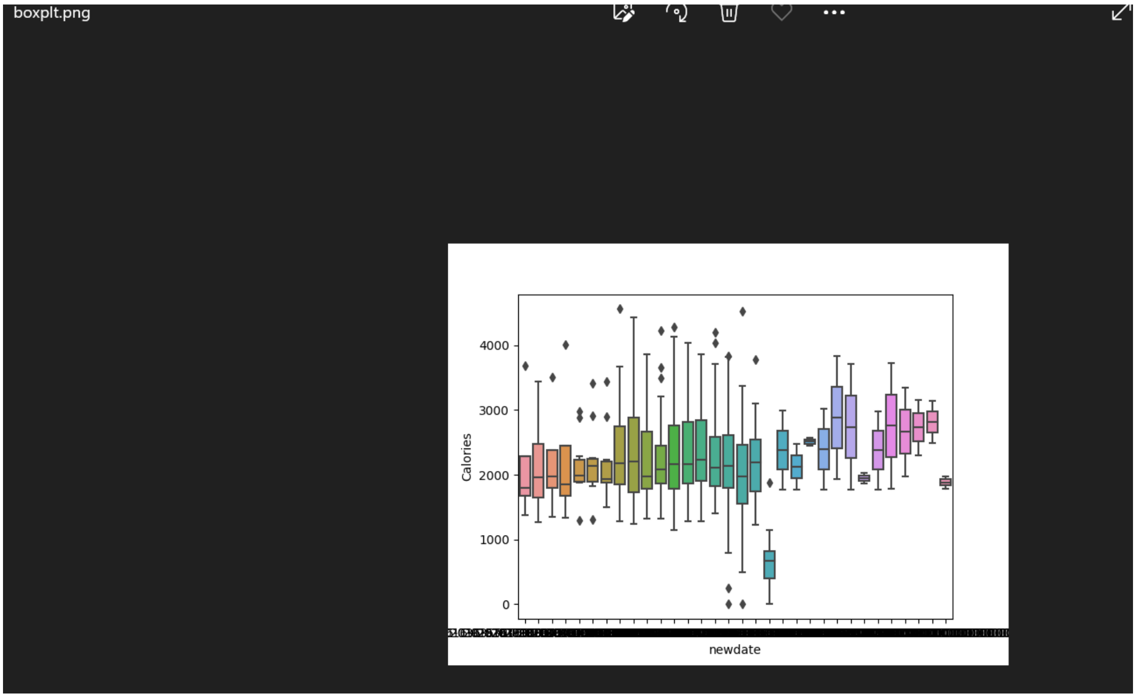
→ `import matplotlib.pyplot as plt`

→ `print(df['newdate'])`

→ `sns.boxplot(x='newdate',y='Calories',data=df)`

→ `plt.savefig('boxplt.png')` To save the graph or plot in to the file, file can be png, pdf,jpg…etc

→

➔ 

➔ Based on the plot make some analysis, such as when people are burning more callories, on what days the people or burning less calorie, if the calories are burnt less on some day you would like to send a notification when you create an app, to them saying you need to burn more calories on so and so day may be monday(as per the data). This will be your one of the pattern identification and suggestion you came up. Similarly you need to find out various patterns when you look in to the data and graphs.

➔ Get the week of the year to understand in what weeks the people burning more or less calories

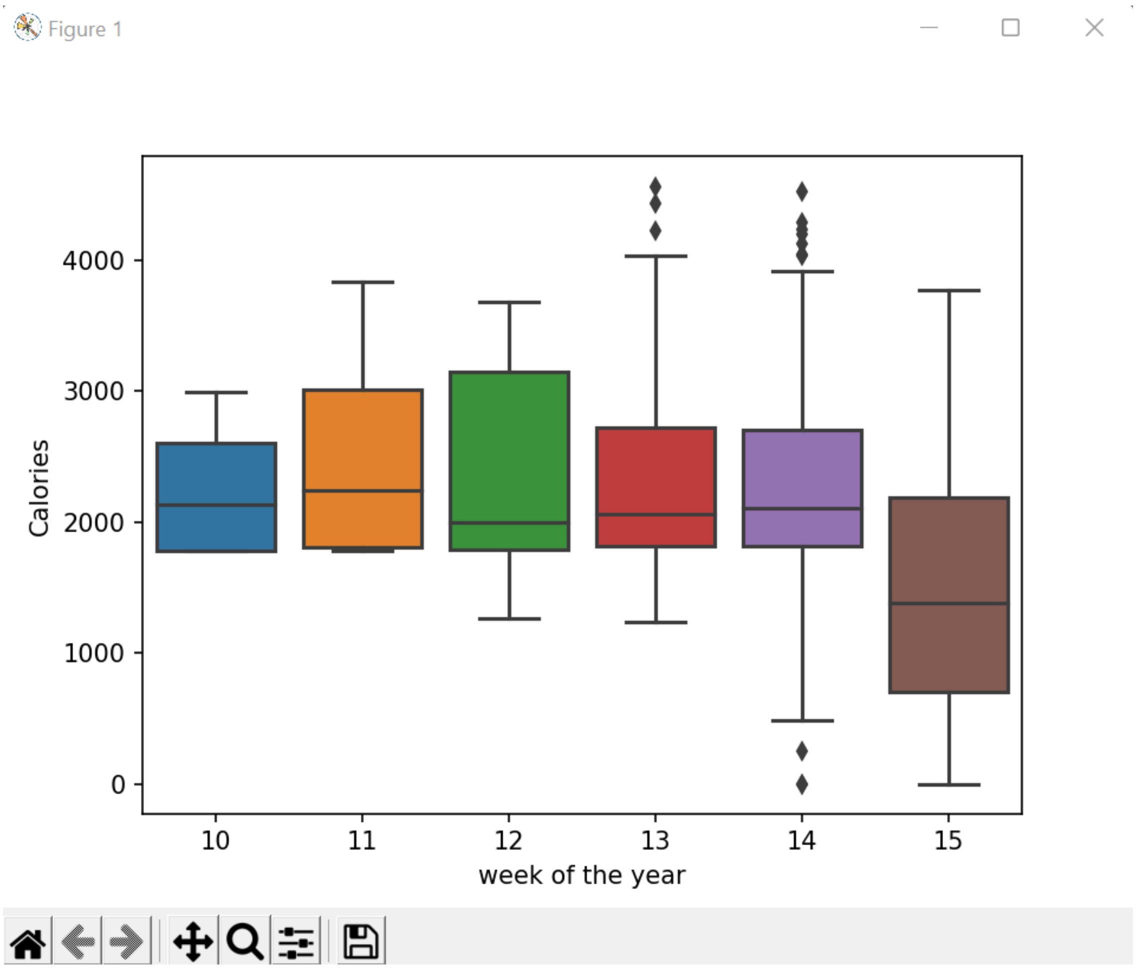
➔ `df['week of the year']=pd.DatetimeIndex(df['ActivityDate']).week`

➔ `print(df[['week of the year','newdate']])`

```
     week of the year    newdate
0                  12 2016-03-25
1                  12 2016-03-26
2                  12 2016-03-27
3                  13 2016-03-28
4                  13 2016-03-29
..                ...        ...
452                14 2016-04-08
453                14 2016-04-09
454                14 2016-04-10
```

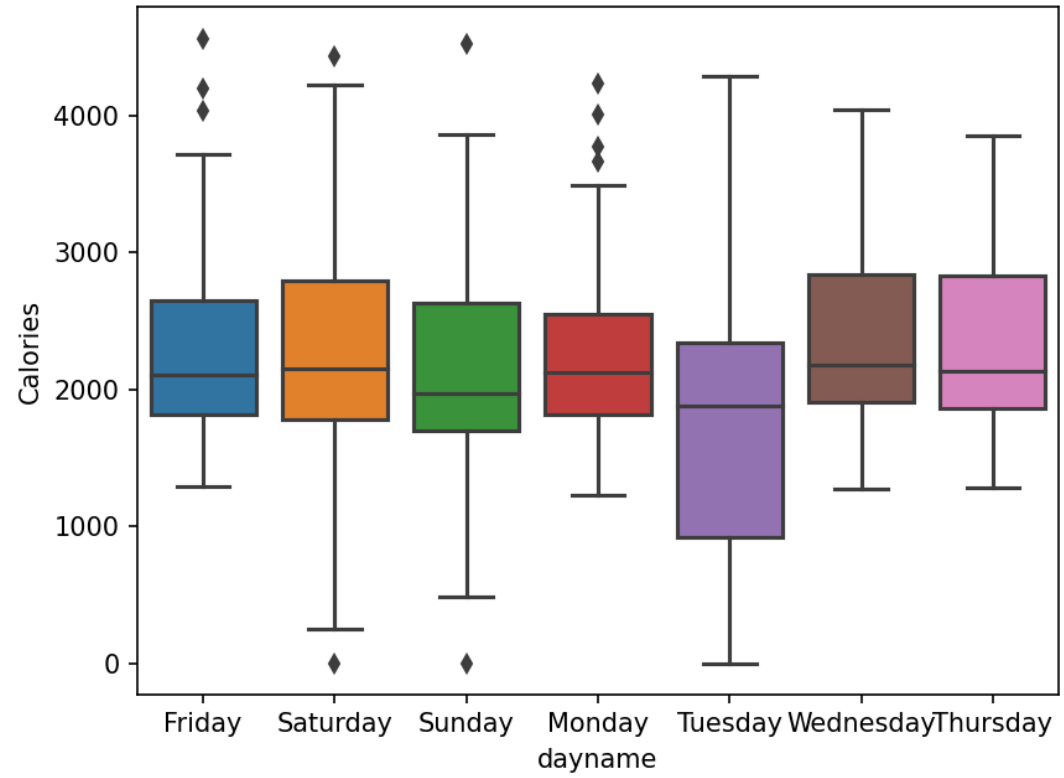
➔ `sns.boxplot(x='week of the year',y='Calories',data=df)`

➔ `plt.show()`



→ Calories versus dayname to understand which day people burn more calories

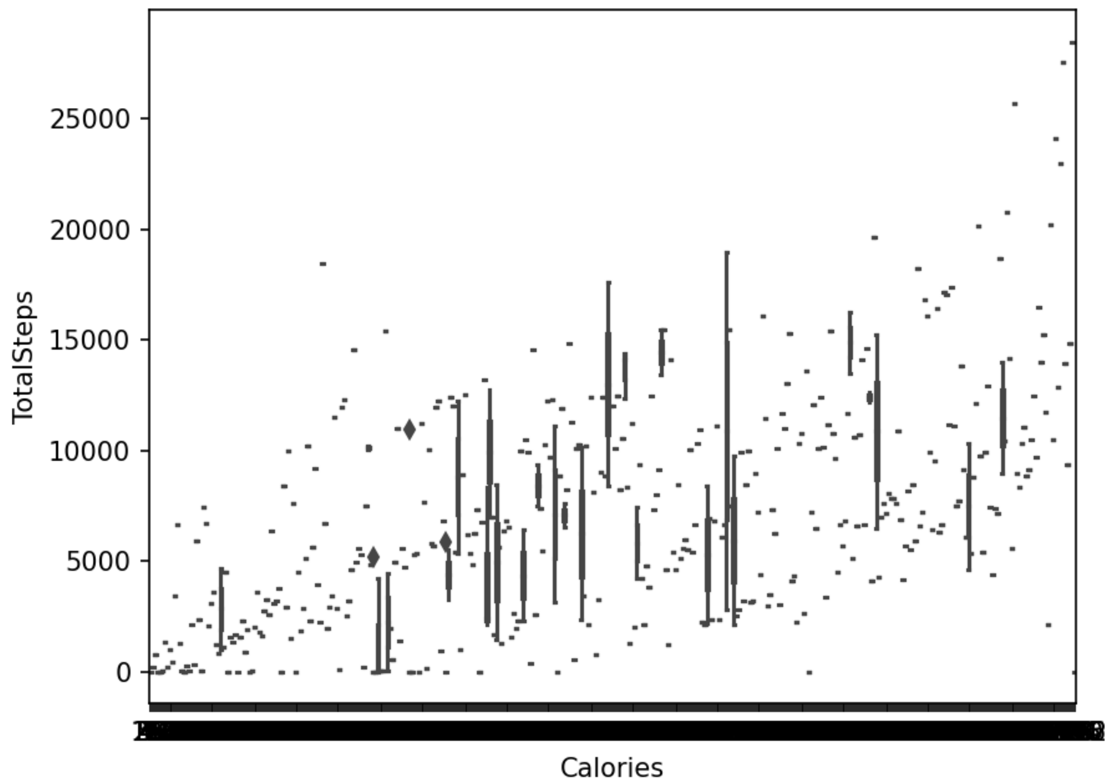
→
```
sns.boxplot(x='dayname',y='Calories',data=df)
```
→
```
plt.show()
```



➔

➔ `sns.boxplot(x='Calories',y='TotalSteps',data=df)`

➔ `plt.show()`



➔

➔ Step2- DATA Transformation or Data Engineering

➔ We will be able to perform Data transformation or data engineering once we have complete data understanding and business understanding.

- Data transformation involves
    - Changing data types from string to datetime type.
    - Deriving new columns from existing columns (as we did for year, month,day..etc)
    - Converting non normal distribution of the data to normal distribution
    - Removing the skewness of the data
    - Deriving the new field based on combination of the existing fields
    - You need to check for the distribution of each and every column in the distribution and need to convert all of them to normal distribution if it is non normal distributed.
    - During this time think of possible patterns and recommendations you can come up with by deriving the new column by using combinations of given columns.
    - Build recommendation system based on longitude and latitude data to suggest them to goto some fitness centre, to some near by shops to buy fitness related accessories.

Goto the [iNeuronai/EDACollection: a collection of different Exploratory Data Analysis Aproaches (github.com)]() for many more open source EDA collections for your EDA practice. Mainly look in to the zomato and FIFA EDA.

General Tasks involved in the EDA

- **Problem Statement**
- **importing necessary libraries.**
- **Loading train dataset** df1 = pd.read_csv('blackFriday_train.csv')
- **Loading test dataset** df2 = pd.read_csv('blackFriday_test.csv'
- **Merging both train and test dataset.** df = df1.append(df2, sort=False)
- df.shape
- **visualizing fist 5 rows of the dataset.** df.head()
- **Describing the basic statistics of the data.** df.describe()
- **Dropping unnecessary fields from the dataset.** df.drop(['User_ID'],axis=1,inplace=True)
- **Converting categorical data into integer ones by using mapping function.** df['Gender']=df['Gender'].map({'F':0, 'M':1})
- df['Gender'].head(10) # *checking the column after tranasformation*
- **visualizing the unique values of the particular field.** df.Age.unique()
- **Mapping the range variable into integer ones.** df['Age']=df['Age'].map({'0-17':1, '18-25':2, '26-35':3, '36-45':4, '46-50':5, '51-55':6, '55+':7 })
- **creating dummies for the categorical data.**city = pd.get_dummies(df['City_Category'],drop_first=True)
- **Concatinaing dummy variables with original dataset.** df = pd.concat([df,city],axis=1)
- **visualizing last 5 rows of the dataset.**df.tail()
- **Checking for columnwise null values** df.isnull().sum()
- Or train.isna().sum()
-
- **visualizing unique values of fields which contains NAN values for different columns.** df.Product_Category_1.unique()
- **Value count of each variable.** df.Product_Category_2.value_counts()
- **Finding mode of the field.** df.Product_Category_1.mode()

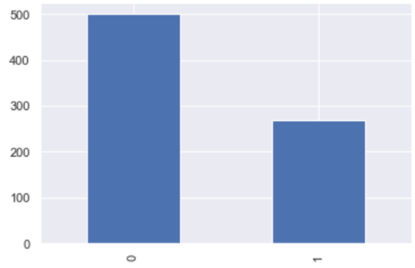
- **Renaming the columns.**
  df.rename(columns={'Product_Category_1':'cat1','Product_Category_2':'cat2', 'Product_Category_3':'cat3'},inplace=True)
- **Looking at the column names after the rename operation.**df.columns
- **filling the nan values with the mode.** df['cat2'] = df['cat2'].fillna(df['cat2'].mode()[0])
- **Filling the nan values with the mean of the column.** df['Purchase'] = df['Purchase'].fillna(df['Purchase'].mean())
- **Rechecking the null values.**df.isnull().sum()
- **Dropping the Column.**df.drop('City_Category',axis=1, inplace=True)
- **Replacing the value by using str method.**
  df['Stay_In_Current_City_Years']=df.Stay_In_Current_City_Years.str.replace('+','') *# replacing + with blank*
- **Checking the allover info of the dataset.**df.info()
- **converting the datatypes into integer ones as the datatype for these columns are shown as unsigned int in the info above** df['B']=df['B'].astype(int)
- **Rechecking the datatypes of the dataset.** df.dtypes
- **Creating a checkpoint.** df_i = df.copy()
- **Visualizing Age Vs Purchased.** sns.barplot('Age','Purchase',hue='Gender',data=df_i)
- **Visualizing Occupation Vs Purchased.**
  sns.barplot('Occupation','Purchase',hue='Stay_In_Current_City_Years',data=df_i)
- **Visualizing Product_category1 Vs Purchased.**
- **Label Encoding the Object type Columns**
-
```
%%time
# Labeling the catagories with integers
for col in train.columns:
    if train[col].dtypes == object: # if the column has categorical values
        l_unique = train[col].unique() # find the unique values
        v_unique = np.arange(len(l_unique)) # create a list of number from zero to the length of the I_unique values
        train[col].replace(to_replace=l_unique, value=v_unique, inplace=True) # replace the categorical values with numerical values
        train[col] = train[col].astype('int') # change the type from int64 to int32

        # same has been done for test data as well
        test[col].replace(to_replace=l_unique, value=v_unique, inplace=True)
        test[col] = test[col].astype('int')
```
- **Splitting the dataset into the Training set and Test set**
- *from sklearn.model_selection import train_test_split*
    - *X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 5)*
-
- **Feature Scaling So that data in all the columns are to the same scale**
- *from sklearn.preprocessing import StandardScaler*
- *sc = StandardScaler()*
- *X_train = sc.fit_transform(X_train)*
    - *X_test = sc.transform(X_test)*
-
- **Now we have features for both training and testing. The data can now be converted to a dataframe, if necessary, and can be fed to a machine learning model.**

Some more EDA tasks list

➔ ***checking the balance of the data by plotting the count of outcomes by their value***

```
## checking the balance of the data by plotting the count of outcomes by their value
color_wheel = {1: "#0392cf",
               2: "#7bc043"}
colors = diabetes_data["Outcome"].map(lambda x: color_wheel.get(x + 1))
print(diabetes_data.Outcome.value_counts())
p=diabetes_data.Outcome.value_counts().plot(kind="bar")
```

```
0    500
1    268
Name: Outcome, dtype: int64
```



The above graph shows that the data is biased towards datapoints having outcome value as 0 where it means that diabetes was not present actually. The number of non-diabetics is almost twice the number of diabetic patients

➔ **Scatter matrix of uncleaned data**

from pandas.plotting import scatter_matrix
p=scatter_matrix(diabetes_data,figsize=(25, 25))

➔ **Pair plot for clean data**

➔ p=sns.pairplot(diabetes_data_copy, hue = 'Outcome')

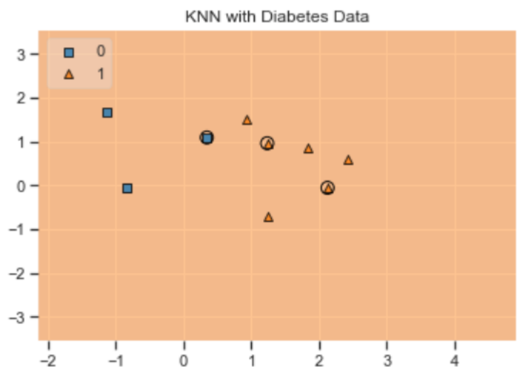
➔ **Heatmap for unclean data**

➔ **Heatmap for unclean data**

- **plt.figure(figsize=(12,10))** ***# on this line I just set the size of figure to 12 by 10.***
- **p=sns.heatmap(diabetes_data.corr(), annot=True,cmap ='RdYlGn')** ***# seaborn has very simple solution for heatmap***

➔ ***trying to plot decision boundary***

```
value = 20000
width = 20000
plot_decision_regions(X.values, y.values, clf=knn, legend=2,
                      filler_feature_values={2: value, 3: value, 4: value, 5: value, 6: value, 7: value},
                      filler_feature_ranges={2: width, 3: width, 4: width, 5: width, 6: width, 7: width},
                      X_highlight=X_test.values)

# Adding axes annotations
#plt.xlabel('sepal length [cm]')
#plt.ylabel('petal length [cm]')
plt.title('KNN with Diabetes Data')
plt.show()
```



➔

➔ ***# Plotiing the wordcloud for the Nationalit column***

- plt.subplots(figsize=(25,15))
- wordcloud = WordCloud(
-                background_color='white',
-                width=1920,
-                height=1080
-               ).generate(" ".join(df.Nationality))
- plt.imshow(wordcloud)
- plt.axis('off')
- plt.savefig('graph.png')
- plt.show()

➔ **Count plot**
- plt.figure(figsize = (18, 8))
- plt.style.use('fivethirtyeight')
- ax = sns.countplot('Position', data = df, palette = 'dark')
- ax.set_xlabel(xlabel = 'Different Positions in Football', fontsize = 16)
- ax.set_ylabel(ylabel = 'Count of Players', fontsize = 16)
- ax.set_title(label = 'Comparison of Positions and Players', fontsize = 20)
- plt.show()

➔ **Dist plot**
- x = df.Special
- plt.figure(figsize = (12, 8))
- plt.style.use('tableau-colorblind10')
-
- ax = sns.distplot(x, bins = 58, kde = False, color = 'cyan')
- ax.set_xlabel(xlabel = 'Special score range', fontsize = 16)
- ax.set_ylabel(ylabel = 'Count of the Players',fontsize = 16)
- ax.set_title(label = 'Histogram for the Speciality Scores of the Players', fontsize = 20)
- plt.show()

➔ **Bar plot**
- plt.rcParams['figure.figsize'] = (15, 7)
- ax = sns.barplot(x = data_countries['Nationality'], y = data_countries['Overall'], palette = 'spring') *# creating a bargraph*
- ax.set_xlabel(xlabel = 'Countries', fontsize = 9)
- ax.set_ylabel(ylabel = 'Overall Scores', fontsize = 9)
- ax.set_title(label = 'Distribution of overall scores of players from different countries', fontsize = 20)
- plt.show()
-

➔ ***# comparing the performance of left-footed and right-footed footballers***

➔ ***# ballcontrol vs dribbing***

➔
- sns.lmplot(x = 'BallControl', y = 'Dribbling', data = data, col = 'Preferred Foot')
- plt.show

➔ ***# defining a method to show the leading performers***
- def graphPolar(id = 0):
-     if 0 <= id < len(data.ID):
-         details(row = players.index[id],
-                 title = players['Name'][id],
-                 age = players['Age'][id],
-                 photo = players['Photo'][id],
-                 nationality = players['Nationality'][id],

- image = players['Flag'][id],
- logo = players['Club_Logo'][id],
- club = players['Club'][id])
- else:
- print('The base has 17917 players. You can put positive numbers from 0 to 17917')
- graphPolar(0)
- 

Some Knowledge on EDA concepts

DataFrame.describe() method generates descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values. This method tells us a lot of things about a dataset. One important thing is that the describe() method deals only with numeric values. It doesn't work with any categorical values. So if there are any categorical values in a column the describe() method will ignore it and display summary for the other columns unless parameter include="all" is passed.

Now, let's understand the statistics that are generated by the describe() method:

- count tells us the number of NoN-empty rows in a feature.
- mean tells us the mean value of that feature.
- std tells us the Standard Deviation Value of that feature.
- min tells us the minimum value of that feature.
- 25%, 50%, and 75% are the percentile/quartile of each features. This quartile information helps us to detect Outliers.
- max tells us the maximum value of that feature.

The Question creeping out of this summary

Can minimum value of below listed columns be zero (0)?
On these columns, a value of zero does not make sense and thus indicates missing value.

Following columns or variables have an invalid zero value:

1. Glucose
2. BloodPressure
3. SkinThickness
4. Insulin
5. BMI

It is better to replace zeros with nan since after that counting them would be easier and zeros need to be replaced with suitable values

Skewness

A ***left-skewed distribution*** has a long left tail. Left-skewed distributions are also called negatively-skewed distributions. That's because there is a long tail in the negative direction on the number line. The mean is also to the left of the peak.

A ***right-skewed distribution*** has a long right tail. Right-skewed distributions are also called positive-skew distributions. That's because there is a long tail in the positive direction on the number line. The mean is also to the right of the peak.

Pearson's Correlation Coefficient: helps you find out the relationship between two quantities. It gives you the measure of the strength of association between two variables. The value of Pearson's Correlation Coefficient can be between -1 to +1. 1 means that they are highly correlated and 0 means no correlation.

A heat map is a two-dimensional representation of information with the help of colors. Heat maps can help the user visualize simple or complex information.

Scaling the data

$$z = \frac{x_i - \mu}{\sigma}$$

data Z is rescaled such that $\mu = 0$ and $\sigma = 1$, and is done through this formula:

Test Train Split and Cross Validation methods

Train Test Split : To have unknown datapoints to test the data rather than testing with the same points with which the model was trained. This helps capture the model performance much better.

Cross Validation: When model is split into training and testing it can be possible that specific type of data point may go entirely into either training or testing portion. This would lead the model to perform poorly. Hence over-fitting and underfitting problems can be well avoided with cross validation techniques
About Stratify : Stratify parameter makes a split so that the proportion of values in the sample produced will be the same as the proportion of values provided to parameter stratify.

For example, if variable y is a binary categorical variable with values 0 and 1 and there are 25% of zeros and 75% of ones, stratify=y will make sure that your random split has 25% of 0's and 75% of 1's.

Model Performance Analysis

1. Confusion Matrix

The confusion matrix is a technique used for summarizing the performance of a classification algorithm i.e. it has binary outputs.

n=165	**Predicted: NO**	**Predicted: YES**	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

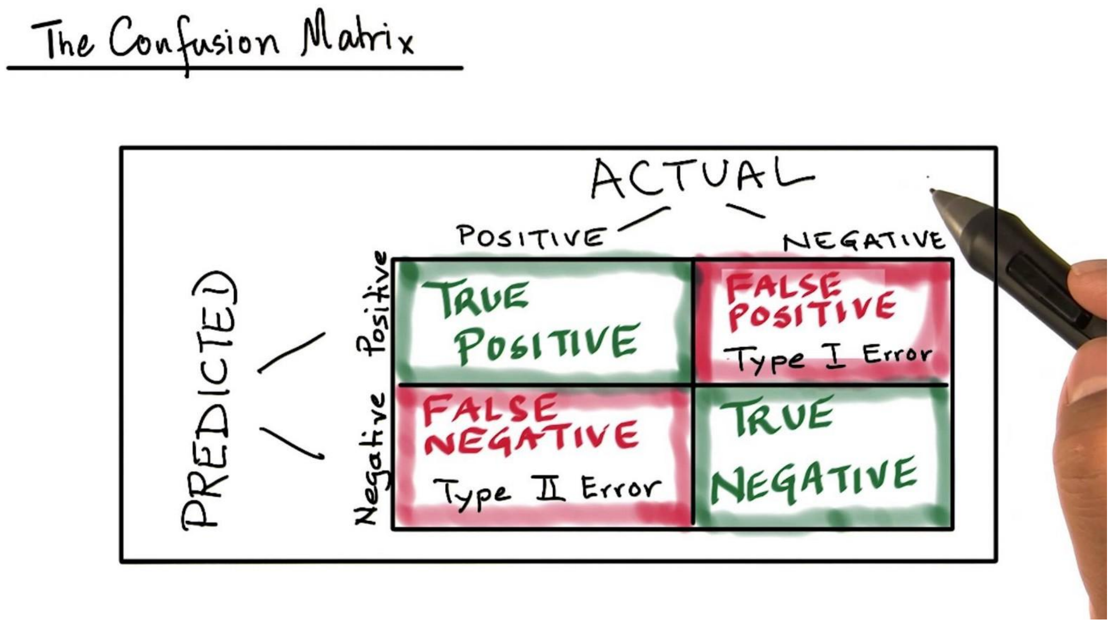
In the famous cancer example:

Cases in which the doctor predicted YES (they have the disease), and they do have the disease will be termed as TRUE POSITIVES (TP). The doctor has correctly predicted that the patient has the disease.

Cases in which the doctor predicted NO (they do not have the disease), and they don't have the disease will be termed as TRUE NEGATIVES (TN). The doctor has correctly predicted that the patient does not have the disease.

Cases in which the doctor predicted YES, and they do not have the disease will be termed as FALSE POSITIVES (FP). Also known as "Type I error".

Cases in which the doctor predicted NO, and they have the disease will be termed as FALSE NEGATIVES (FN). Also known as "Type II error".



2. Classification Report

Report which includes Precision, Recall and F1-Score.

Precision Score

TP – True Positives

FP – False Positives

Precision – Accuracy of positive predictions.

Precision = TP/(TP + FP)

Recall Score

FN – False Negatives

Recall(sensitivity or true positive rate): Fraction of positives that were correctly identified.

Recall = TP/(TP+FN)

F1 Score

F1 Score (aka F-Score or F-Measure) – A helpful metric for comparing two classifiers.

F1 Score takes into account precision and the recall.

It is created by finding the the harmonic mean of precision and recall.

F1 = 2 x (precision x recall)/(precision + recall)

Precision - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all passengers that labeled as survived, how many actually survived? High precision relates to the low false positive rate. We have got 0.788 precision which is pretty good.

Precision = TP/TP+FP

***Recall (Sensitivity)* -** Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes. The question recall answers is: Of all the passengers that truly survived, how many did we label? A recall greater than 0.5 is good.

Recall = TP/TP+FN

***F1 score* -** F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall.

F1 Score = 2*(Recall* Precision) / (Recall + Precision)

. ROC - AUC

ROC (Receiver Operating Characteristic) Curve tells us about how good the model can distinguish between two things (e.g If a patient has a disease or no). Better models can accurately distinguish between the two. Whereas, a poor model will have difficulties in distinguishing between the two

Well Explained in this video: https://www.youtube.com/watch?v=OAl6eAyP-yo

Hyper Parameter optimization

Grid search is an approach to hyperparameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid.

Let's consider the following example:

Suppose, a machine learning model X takes hyperparameters a1, a2 and a3. In grid searching, you first define the range of values for each of the hyperparameters a1,

a2 and a3. You can think of this as an array of values for each of the hyperparameters. Now the grid search technique will construct many versions of X with all the possible combinations of hyperparameter (a1, a2 and a3) values that you defined in the first place. This range of hyperparameter values is referred to as the grid.

Suppose, you defined the grid as: a1 = [0,1,2,3,4,5] a2 = [10,20,30,40,5,60] a3 = [105,105,110,115,120,125]

Note that, the array of values of that you are defining for the hyperparameters has to be legitimate in a sense that you cannot supply Floating type values to the array if the hyperparameter only takes Integer values.

Now, grid search will begin its process of constructing several versions of X with the grid that you just defined.

It will start with the combination of [0,10,105], and it will end with [5,60,125]. It will go through all the intermediate combinations between these two which makes grid search computationally very expensive.