Introduction to Data Science in Python

Getting Started with Python

We can use python to visualize, interpret, mutate and create data. Let's start by learning how to import specific modules of python for data science.

Modules (sometimes called packages or libraries) help group together related sets of tools in Python. We can import the python modules in two ways.

1. Without alias
2. With alias

Alias is basically importing the module and then rferencing that module to something else. This technique is generally used to name the modules shortly. For example, pandas as pd, statsmodel as sm and seaborn as sns.

Each module has a standard alias, which allows you to access the tools inside of the module without typing as many characters. For example, aliasing lets us shorten seaborn.scatterplot() to sns.scatterplot().

In [1]:

```
# Use an import statement to import statsmodels without alias
import statsmodels

# Import statsmodels under the alias sm
import statsmodels as sm

# Use an import statement to import seaborn with alias sns
import seaborn as sns

# Use an import statement to import pandas with alias pd
import pandas as pd

# Use an import statement to import numpy with alias np
import numpy as np
```

We can create variables in python. There are different data types in python like Strings, Integer, Floats. A string represents text. A string is surrounded by quotation marks **(' or ")** and can contain letters, numbers, and special characters. It doesn't matter if you use single (') or double (") quotes, but it's important to be consistent throughout our code.

There are some naming conventions variable in python. The most importants are:

- There can't be any space in between rather we can use **_**
- The variable name can't start with numbers
- We can't use - in between the variable name
- usually starts with small letter

In [2]:

```
# Bayes' favorite toy
favorite_toy = "Mr. Squeaky"

# Bayes' owner
owner = 'DataCamp'


# Display variables
print(favorite_toy)
print(owner)
```

```
Mr. Squeaky
DataCamp
```

In python we can create function or use any existing function from a module. A function is an action which take some inputs and gives us an output. Specific functions performs specific actions.

A function can take some arguments depending on the function. The types of arguments are:

1. Positional Arguments
2. Keyword Argument

Anatomy of a Function

Function Name:

- Starts with the module that the function "lives" in the module
- Followed by the name of the function
- Function name is always followed by the parantheses **()**

Positional Arguments

- These are inputs to a function; they tell the function how to do it's job
- Order of the arguments matters!

Keyword Argument

- Must come after the positional argument
- Start with the name of the argument (), than an equals sign(=)
- Followed by the argument

Most Common Errors while writing function

- Missing commas between arguments
- Missing closed paranthesis

We'll load the data into a DataFrame, a special data type from the pandas module. It represents spreadsheet-like data (something with rows and columns).

We can create a DataFrame from a CSV (comma-separated value) file by using the function pd.read_csv().

In [3]:
```
# Load the 'ransom.csv' into a DataFrame
r = pd.read_csv('Advertising.csv')

# Display DataFrame
print(r)
```

```
        TV  radio  newspaper  sales
0    230.1   37.8       69.2   22.1
1     44.5   39.3       45.1   10.4
2     17.2   45.9       69.3    9.3
3    151.5   41.3       58.5   18.5
4    180.8   10.8       58.4   12.9
..     ...    ...        ...    ...
195   38.2    3.7       13.8    7.6
196   94.2    4.9        8.1    9.7
197  177.0    9.3        6.4   12.8
198  283.6   42.0       66.2   25.5
199  232.1    8.6        8.7   13.4

[200 rows x 4 columns]
```

Loading Data in pandas

Panda is a powerful Python libary. Pandas lets you read, modify, and search tabular datasets (like spreadsheets and database tables). We can read csv format files using the **.read_csv('dataset.csv')** function. Let's load a csv file now.

In [4]:
```
foot = pd.read_csv('FootHeight.csv')
```

We can use the function head() to only see the first 5 rows or observation of the dataset

In [5]:
```
print(foot.head())
```

```
   footlength  height
0        32.0      74
1        24.0      66
2        29.0      77
3        30.0      67
4        24.0      56
```

We can also get the info about the dataset and all the variables using the info() function. We can get the data types of the variables and also the number of observations for the variables.

In [6]:
```
print(foot.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   footlength  20 non-null     float64
 1   height      20 non-null     int64  
dtypes: float64(1), int64(1)
memory usage: 448.0 bytes
None
```

Sometimes while working with data it is important that we know how to select something from the data. There are sometimes some irrelevant information or observation in the dataset which we have to get rid of. We can select specific observation or column from a dataset.

Selecting columns

In [7]:

```
credit = pd.read_csv("Credit.csv")
print(credit.head())
print(credit.info())
Ethnicity = credit["Ethnicity"]
print(Ethnicity)
Gender = credit.Gender
print(Gender)
```

```
    Income  Limit  Rating  Cards  Age  Education  Gender Student Married  \
0   14.891   3606     283      2   34         11    Male      No     Yes   
1  106.025   6645     483      3   82         15  Female     Yes     Yes   
2  104.593   7075     514      4   71         11    Male      No      No   
3  148.924   9504     681      3   36         11  Female      No      No   
4   55.882   4897     357      2   68         16    Male      No     Yes   

   Ethnicity  Balance  
0  Caucasian      333  
1      Asian      903  
2      Asian      580  
3      Asian      964  
4  Caucasian      331  
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 11 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Income     400 non-null    float64
 1   Limit      400 non-null    int64  
 2   Rating     400 non-null    int64  
 3   Cards      400 non-null    int64  
 4   Age        400 non-null    int64  
 5   Education  400 non-null    int64  
 6   Gender     400 non-null    object 
```

```
 7   Student    400 non-null    object
 8   Married    400 non-null    object
 9   Ethnicity  400 non-null    object
 10  Balance    400 non-null    int64
dtypes: float64(1), int64(6), object(4)
memory usage: 34.5+ KB
None
0             Caucasian
1                 Asian
2                 Asian
3                 Asian
4             Caucasian
             ...
395           Caucasian
396    African American
397           Caucasian
398           Caucasian
399               Asian
Name: Ethnicity, Length: 400, dtype: object
0        Male
1      Female
2        Male
3      Female
4        Male
        ...
395      Male
396      Male
397    Female
398      Male
399    Female
Name: Gender, Length: 400, dtype: object
```

So we saw that we can select a colum using the **[]** and the string which is the column name. We can also select a column using the dot method. There are some of the common mistakes while we select. These are :-

- We shouldn't forget using the square bracket
- Using the ""
- naming exactly like the dataset column name (case sensitive)

Selecting rows with logic

We can use logical statements in python by using the logical operators. We can also select rows or observation from the datset using the logical statements.

Types of logic in python:

1. **==** equal to
2. **!=** not equal to
3. greater than
4. < smaller than
5. => greater than or equal to
6. => smaller than or equal to

In [8]:
```
price = 2.05
solution = 2+0.05
price == solution
```

Out[8]:
```
True
```

In [9]:
```
#we can see that python is case sensitive
name= "data"
name2 = "Data"
name==name2
```

Out[9]:
```
False
```

We can use logic with dataframes to get the rows corresponding to our logic .

In [10]:
```
credit.Income > 100
```

Out[10]:
```
0      False
1       True
2       True
3       True
4      False
       ...
395    False
396    False
397    False
398    False
399    False
Name: Income, Length: 400, dtype: bool
```

We can use this logic to select the columns from the dataframe corresponding to the logical statement. We can use **[]** and put the logical statement inside and select the column.

In [11]:

```
credit[credit.Income>150]
```

Out[11]:

	Income	Limit	Rating	Cards	Age	Education	Gender	Student	Married	Ethnicity	Ba
28	186.634	13414	949	2	41	14	Female	No	Yes	African American	
85	152.298	12066	828	4	41	12	Female	No	Yes	Asian	
184	158.889	11589	805	1	62	17	Female	No	Yes	Caucasian	
209	151.947	9156	642	2	91	11	Female	No	Yes	African American	
261	180.379	9310	665	3	67	8	Female	Yes	Yes	Asian	
275	163.329	8732	636	3	50	14	Male	No	Yes	Caucasian	
323	182.728	13913	982	4	98	17	Male	No	Yes	Caucasian	
347	160.231	10748	754	2	69	17	Male	No	No	Caucasian	
355	180.682	11966	832	2	58	8	Female	No	Yes	African American	

In [12]:

```
only_cauc = credit[credit.Ethnicity == 'Caucasian']
print(only_cauc.head())
```

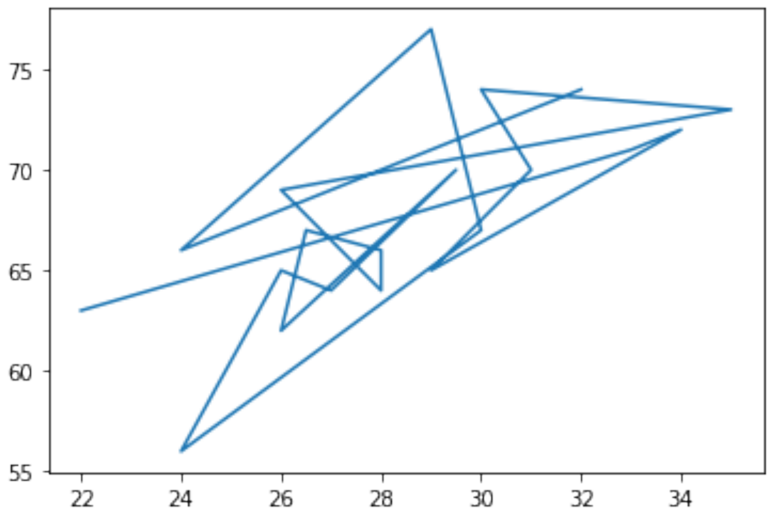
```
     Income  Limit  Rating  Cards  Age  Education  Gender Student Married  \
0    14.891   3606     283      2   34         11    Male      No     Yes
4    55.882   4897     357      2   68         16    Male      No     Yes
5    80.180   8047     569      4   77         10    Male      No      No
8    15.125   3300     266      5   66         13  Female      No      No
10   63.095   8117     589      4   30         14    Male      No     Yes

    Ethnicity  Balance
0   Caucasian      333
4   Caucasian      331
5   Caucasian     1151
8   Caucasian      279
10  Caucasian     1407
```

Plotting with Pyplot

Creating Lineplot

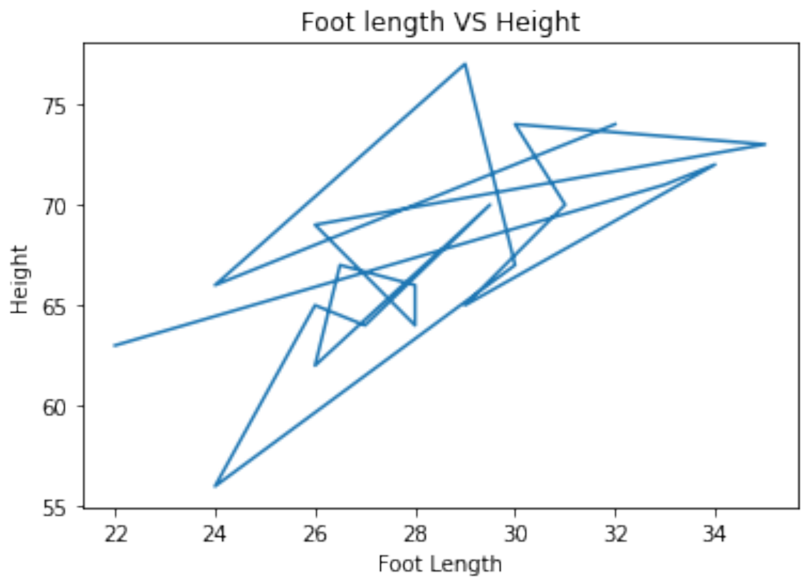
We can use the pyplot module from matplotlib to create simple graph in python. We use the function **.plot** for plotting and **.show** to display the graph. Now let's create a line plot from the dataset we loaded in our workspace before.

In [13]:
```
foot = pd.read_csv('FootHeight.csv')

from matplotlib import pyplot as plt
plt.plot(foot.footlength,foot.height)
plt.show()
```



We can add labels to our graph and also titles using **plt**. We can use the plt.xlabel() and plt.ylabel() for the axis lables and plt.title() for the title of the graph. We can use this functions for a graph anytime before the plt.show() and after the plotting of the graph.

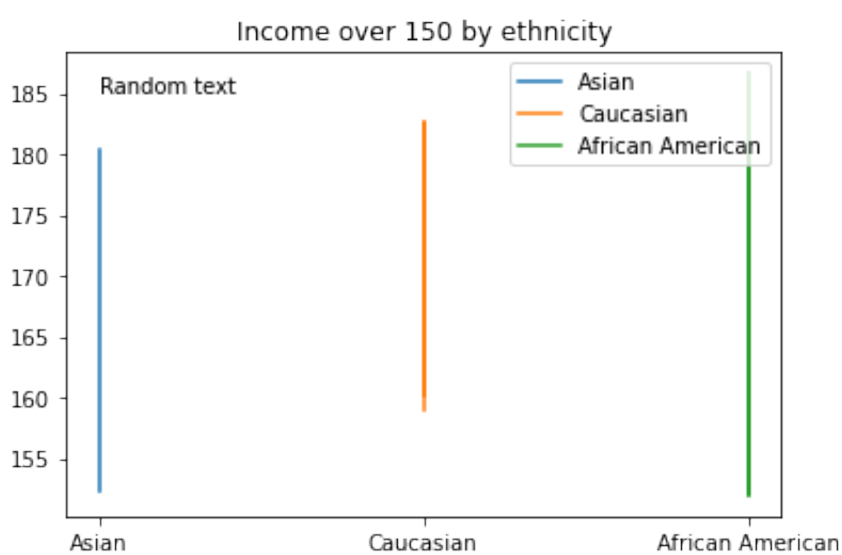
In [14]:
```
plt.plot(foot.footlength,foot.height)
plt.xlabel("Foot Length")
plt.ylabel("Height")
plt.title("Foot length VS Height")
plt.show()
```



We can use annotation in the graph using the plt.text(). This function takes the argument x and y values of the text in the graph and then the text. We can also use legends using the plt.legend() which also takes the argument of **loc=** to specify the location of the legend

In [15]:

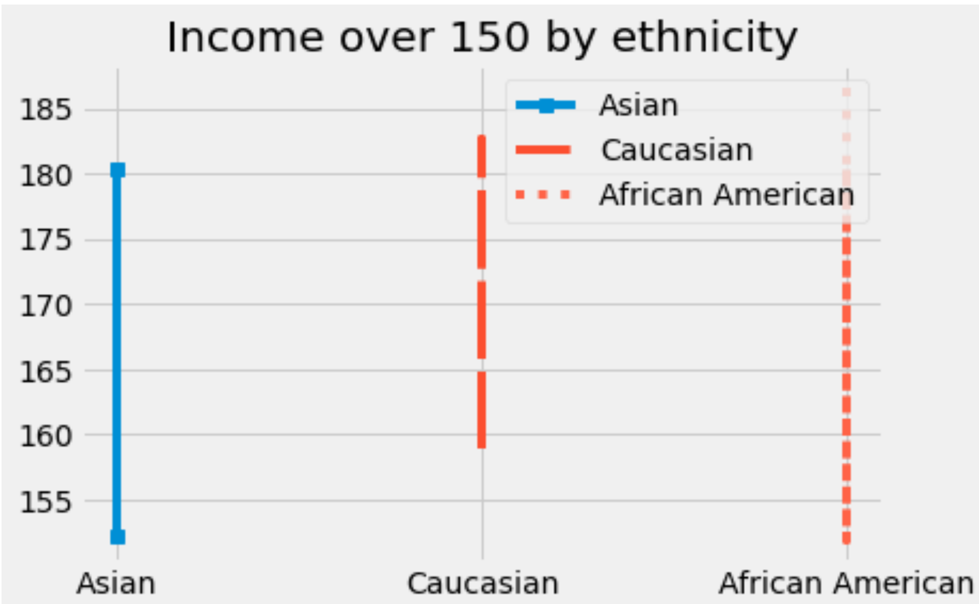
```
Incomeover = credit[credit.Income>150]
Incomeover_asian = Incomeover[Incomeover.Ethnicity =='Asian']
Incomeover_cauc = Incomeover[Incomeover.Ethnicity =='Caucasian']
Incomeover_afam = Incomeover[Incomeover.Ethnicity =='African American']
plt.plot(Incomeover_asian.Ethnicity, Incomeover_asian.Income, label ='Asian')
plt.plot(Incomeover_cauc.Ethnicity, Incomeover_cauc.Income, label ='Caucasian
plt.plot(Incomeover_afam.Ethnicity, Incomeover_afam.Income, label ='African A
plt.title('Income over 150 by ethnicity')
plt.legend(loc='upper right')
plt.text('Asian',185,'Random text')
plt.show()
```



We can do a lot of styling using different arguments in python. We can change the color using the **Color=''** argument. We can cahnge the linetype using the **linestyle=''** . We can also change the line width using the **linewidth=''** argument. We can also use different theme using the **plt.style.use('')** and we have to set that any other plotting code.

In [16]:

```
plt.style.use('fivethirtyeight')
plt.plot(Incomeover_asian.Ethnicity, Incomeover_asian.Income, label ='Asian',
plt.plot(Incomeover_cauc.Ethnicity, Incomeover_cauc.Income,label ='Caucasian'
plt.plot(Incomeover_afam.Ethnicity, Incomeover_afam.Income,label ='African Am
plt.title('Income over 150 by ethnicity')
plt.legend(loc='upper right')
plt.show()
```



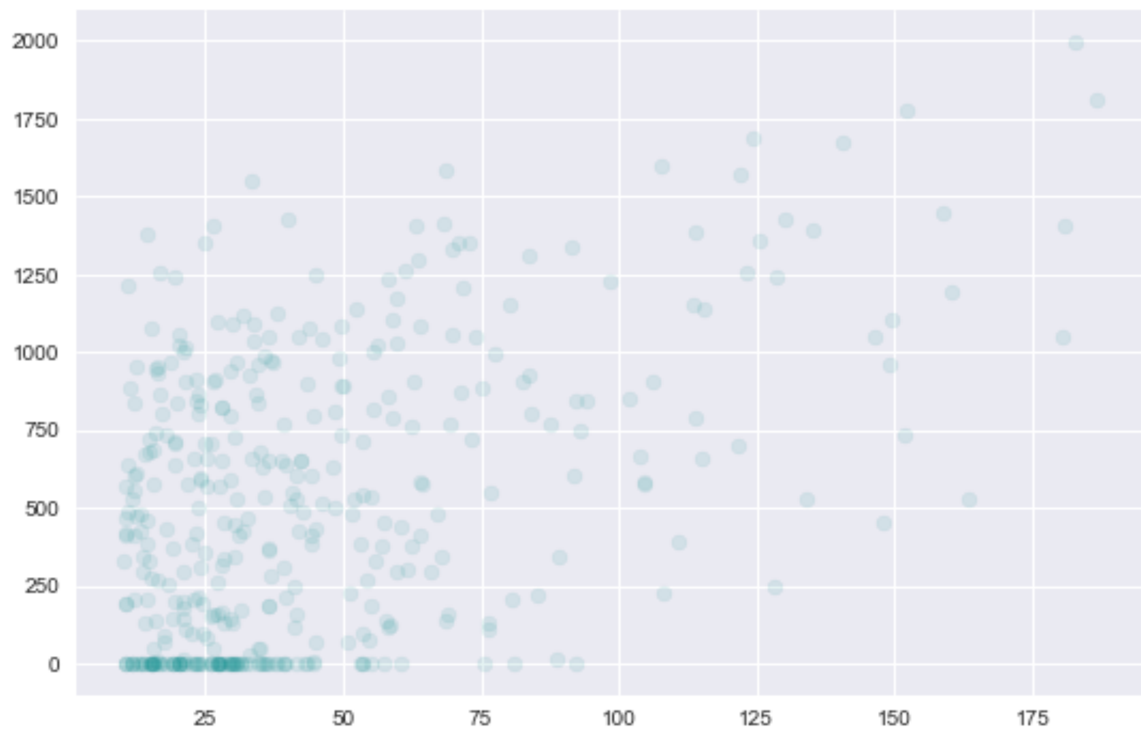
Now let's see how we can create scatterplot in python.

Creating Scatter Plot

We can create scatter plot just like the line plot but using plt.scatter() instead. This also takes arguments like color and marker. We can also use the **alpha=** argument to specify the transparency of the data points.

In [17]:

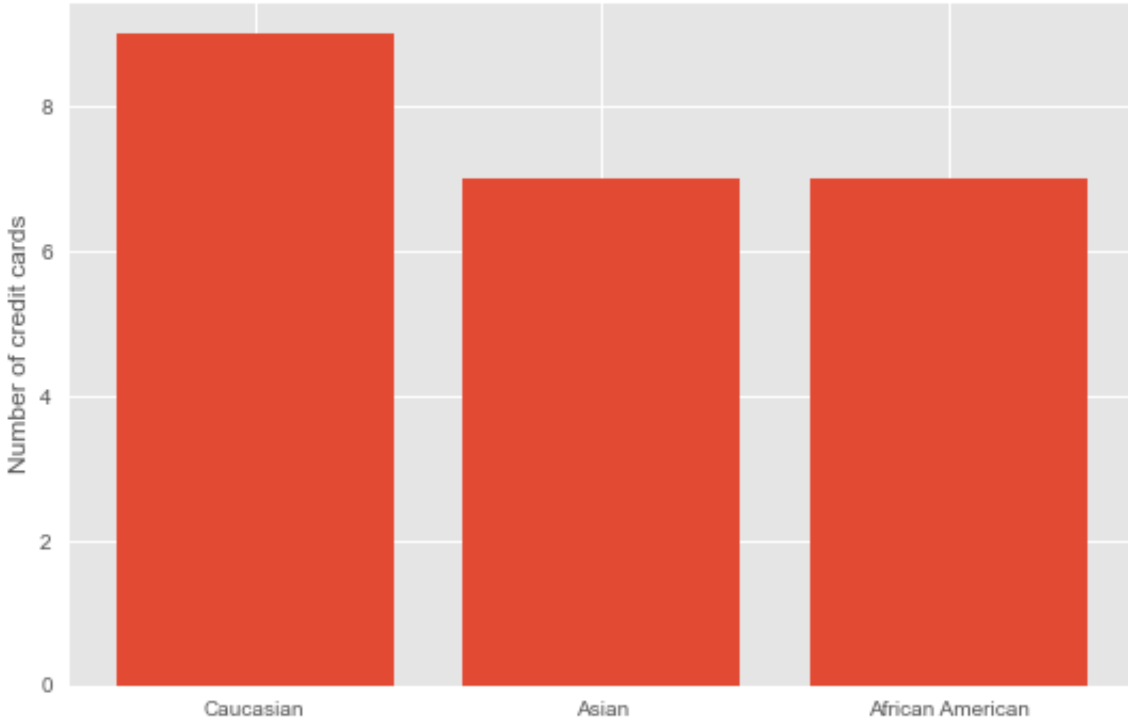
```
plt.style.use("seaborn")
plt.scatter(credit.Income,credit.Balance, color ='DarkCyan', marker = 'o' , a
plt.show()
```



Creating Bar Plot

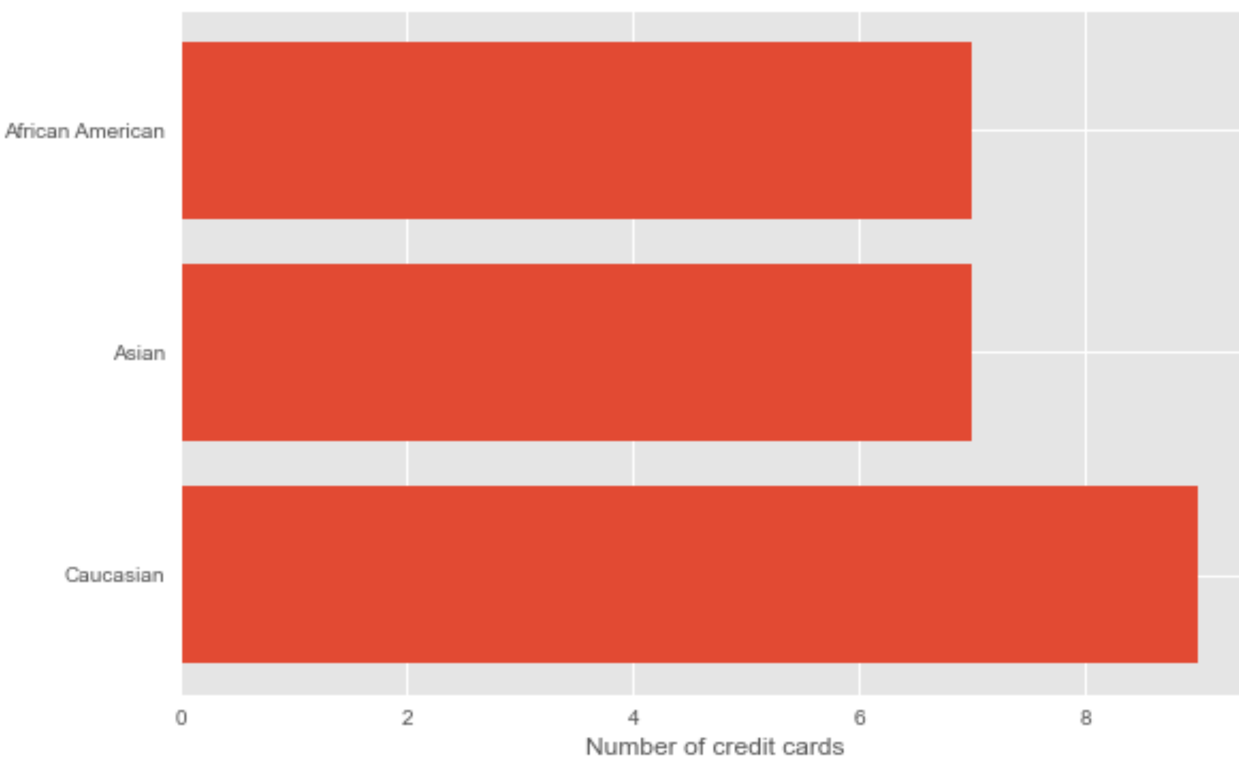
We can create barplot just using the same way but using the plt.bar() we can also use the plt.barh() for the horizontal bar graph.

In [18]:
```
plt.style.use("ggplot")
plt.bar(credit.Ethnicity,credit.Cards)
plt.ylabel("Number of credit cards")
plt.show()
```



In [19]:

```
plt.style.use("ggplot")
plt.barh(credit.Ethnicity,credit.Cards)
plt.xlabel("Number of credit cards")
plt.show()
```

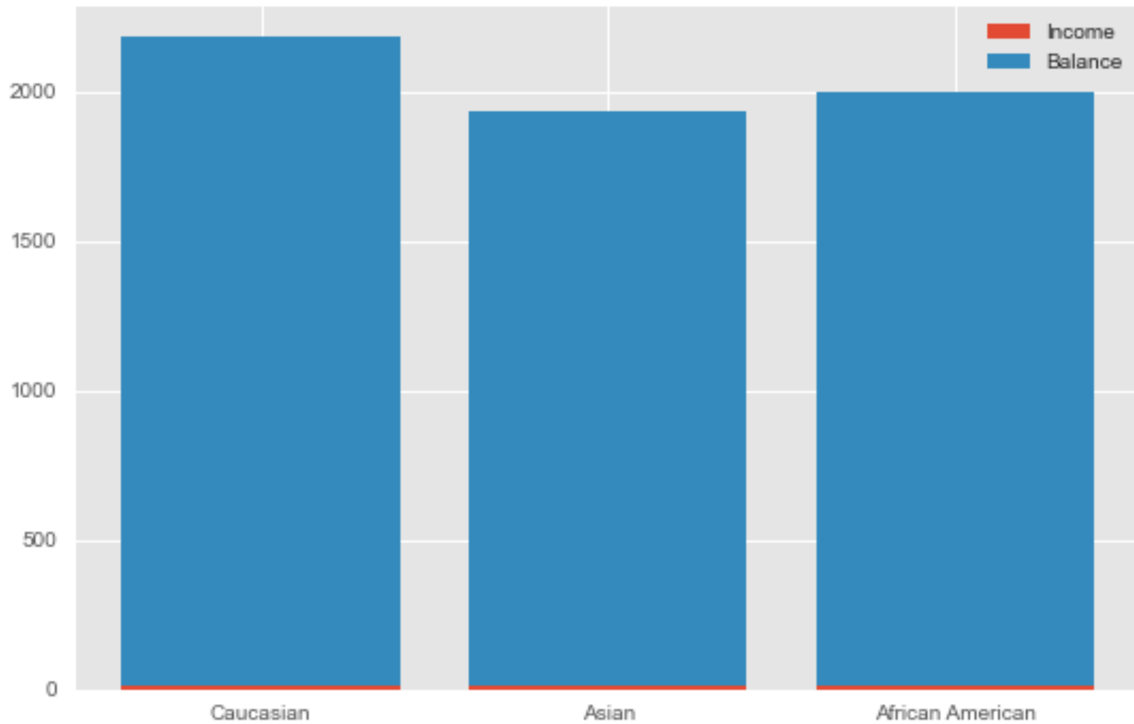


We can also show the error using the yerr= or the xerr= as an argument.

We can also create a stacked bar graph in python.

In [20]:

```
plt.bar(credit.Ethnicity,credit.Income, label ='Income')
plt.bar(credit.Ethnicity,credit.Balance,bottom =credit.Income, label ='Balanc
plt.legend()
plt.show()
```

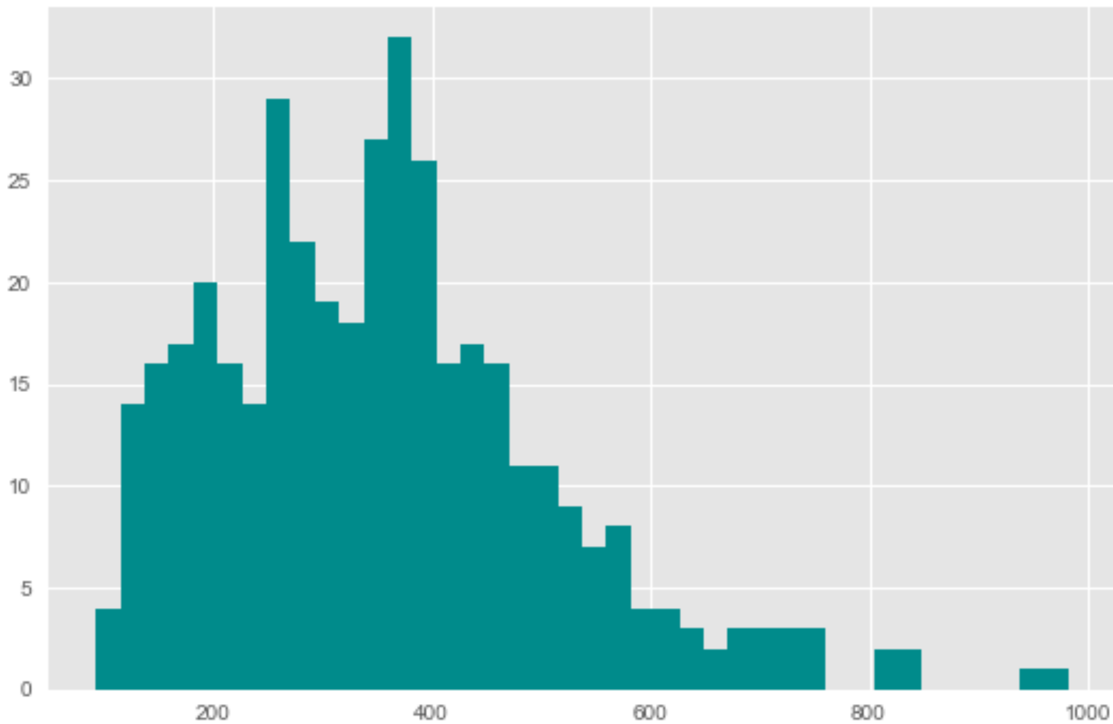


Creating Histogram

We can create histogram in python using the plt.hist() but we have to keep in mind that histograms take only one variable in x axis. We can use the **bins=nbins** argument to cahnge the bins of the histogram. We can also set the range of the histogram using the **range=(xmin,xmax)** fucntion.

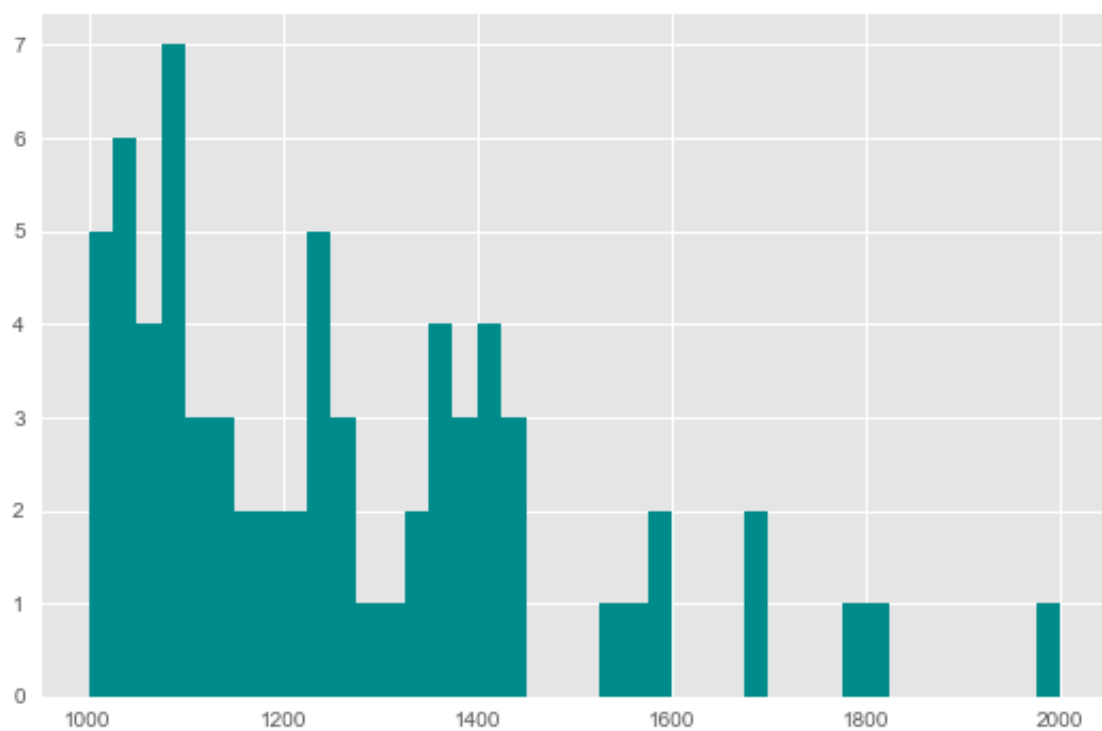
In [21]:

```
plt.hist(credit.Rating, bins=40, color= "DarkCyan")
plt.show()
```



In [22]:

```
plt.hist(credit.Balance,range =(1000,2000), bins=40, color= "DarkCyan")
plt.show()
```



Problem statement

Build a simple linear regression model to predict the Salary Hike using Years of Experience.

Start by Importing necessary libraries

necessary libraries are pandas, NumPy to work with data frames, matplotlib, seaborn for visualizations, and sklearn, statsmodels to build regression models.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from scipy import stats
from scipy.stats import probplot
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn import preprocessing
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

Once, we are done with importing libraries, we create a pandas dataframe from CSV file

In [32]:

```
df = pd.read_csv (r"/Users/rith/Desktop/Python /SLR/Salary_Data.csv")
df.head(n=6)
```

Out[32]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0
5	2.9	56642.0

Perform EDA (Exploratory Data Analysis)

The basic steps of EDA are:

Understand the dataset

-Identifying the number of features or columns -Identifying the features or columns -Identify the size of the dataset -Identifying the data types of features -Checking if the dataset has empty cells -Identifying the number of empty cells by features or columns

-Handling Missing Values and Outliers -Encoding Categorical variables -Graphical Univariate Analysis, Bivariate -Normalization and Scaling

In [33]:
```
len(df.columns) # identify the number of features
```
Out[33]:
```
2
```

In [34]:
```
df.columns # idenfity the features
```
Out[34]:
```
Index(['YearsExperience', 'Salary'], dtype='object')
```

In [35]:
```
df.shape # identify the size of of the dataset
```
Out[35]:
```
(30, 2)
```

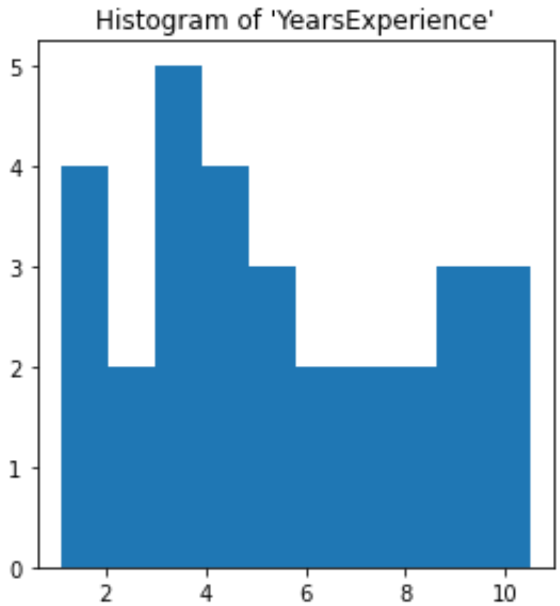
In [36]:
```
df.dtypes # identify the datatypes of the features
```
Out[36]:
```
YearsExperience    float64
Salary             float64
dtype: object
```

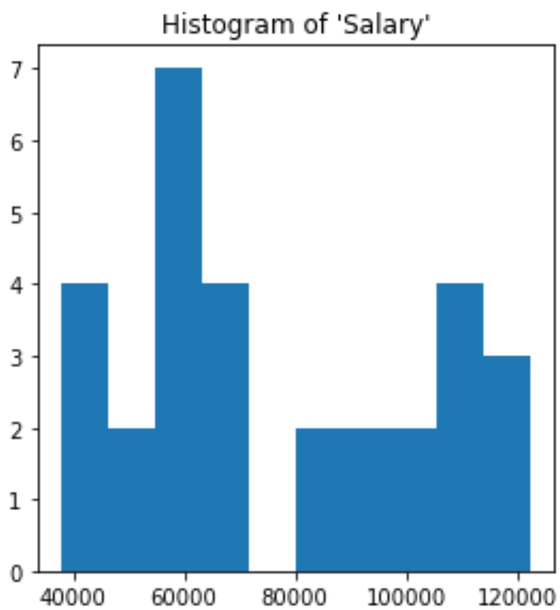
In [37]:
```
df.isnull().values.any() # checking if dataset has empty cells
```
Out[37]:
```
False
```

In [38]:
```
df.isnull().sum() # identify the number of empty cells
```
Out[38]:
```
YearsExperience    0
Salary             0
dtype: int64
```

In [7]:
```
# Histogram
# We can use either plt.hist or sns.histplot
plt.figure(figsize=(20,10))
plt.subplot(2,4,1)
plt.hist(df['YearsExperience'], density=False)
plt.title("Histogram of 'YearsExperience'")
plt.subplot(2,4,5)
plt.hist(df['Salary'], density=False)
plt.title("Histogram of 'Salary'")
```

Out[7]:
```
Text(0.5, 1.0, "Histogram of 'Salary'")
```





In [ ]:

In [ ]:

Our dataset has two columns: YearsExperience, Salary. And both are of float datatype. We have 30 records and no null-values or outliers in our dataset.

Graphical Univariate analysis

For univariate analysis, we have Histogram, density plot, boxplot or violinplot, and Normal Q-Q plot. They help us understand the distribution of the data points and the presence of outliers.

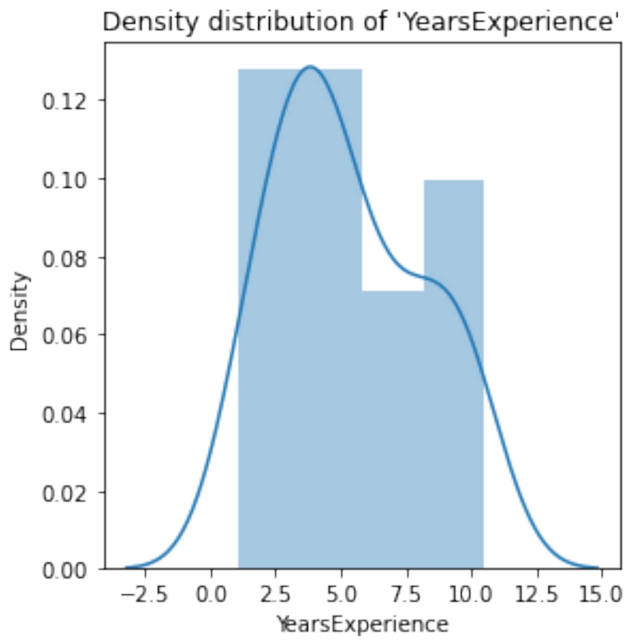
A violin plot is a method of plotting numeric data. It is similar to a box plot, with the addition of a rotated kernel density plot on each side.

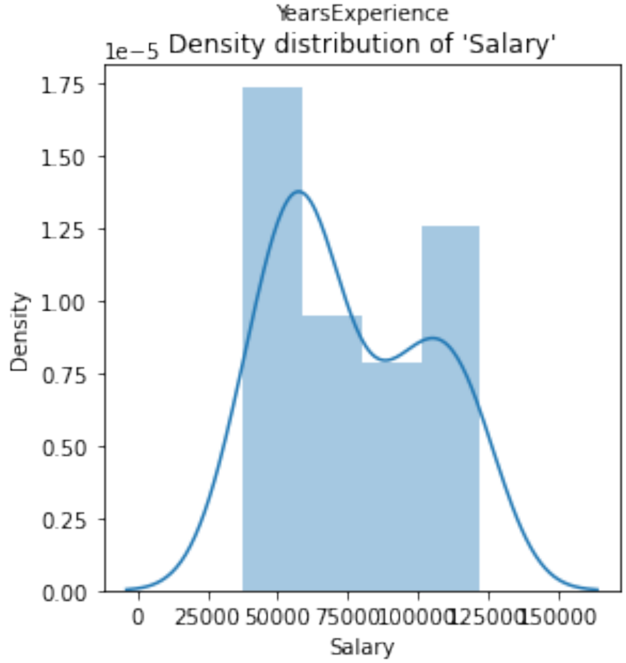
In [9]:

```
# Density plot
plt.figure(figsize=(20,10))
plt.subplot(2,4,2)
sns.distplot(df['YearsExperience'], kde=True)
plt.title("Density distribution of 'YearsExperience'")
plt.subplot(2,4,6)
sns.distplot(df['Salary'], kde=True)
plt.title("Density distribution of 'Salary'")
```

```
/Users/Code/opt/anaconda3/lib/python3.9/site-packages/seaborn/distributions.py
:2619: FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a figure-
level function with similar flexibility) or `histplot` (an axes-level function
for histograms).
  warnings.warn(msg, FutureWarning)
/Users/Code/opt/anaconda3/lib/python3.9/site-packages/seaborn/distributions.py
:2619: FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a figure-
level function with similar flexibility) or `histplot` (an axes-level function
for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[9]: Text(0.5, 1.0, "Density distribution of 'Salary'")



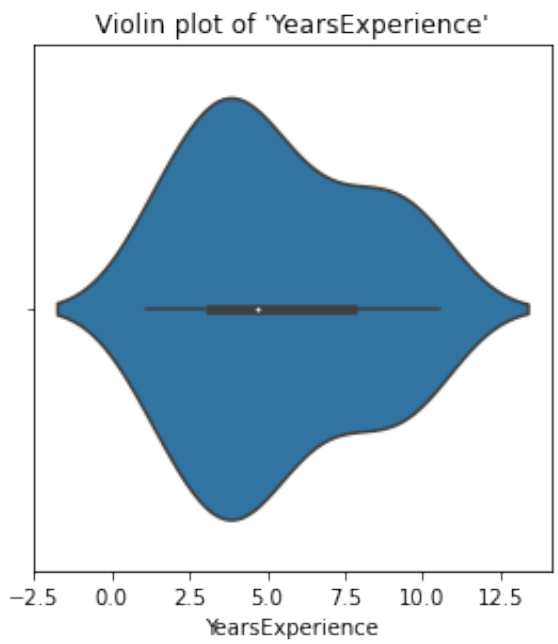


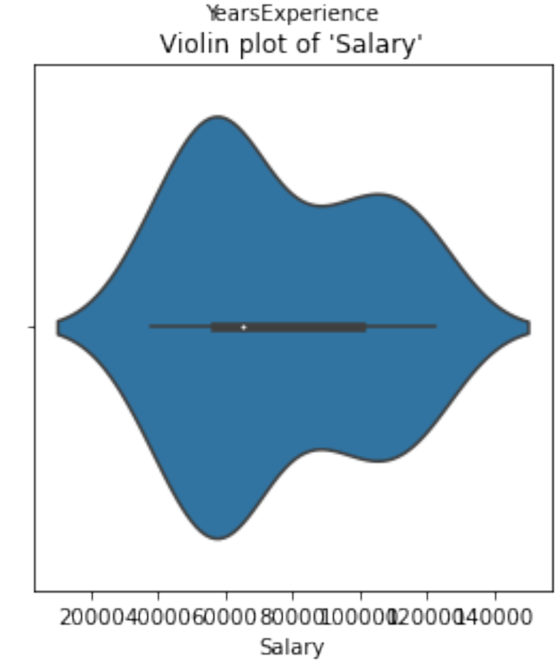
In [11]:

```
# boxplot or violin plot
# A violin plot is a method of plotting numeric data. It is similar to a box
# with the addition of a rotated kernel density plot on each side
plt.figure(figsize=(20,10))
plt.subplot(2,4,3)
# plt.boxplot(df['YearsExperience'])
sns.violinplot(df['YearsExperience'])
# plt.title("Boxlpot of 'YearsExperience'")
plt.title("Violin plot of 'YearsExperience'")
plt.subplot(2,4,7)
# plt.boxplot(df['Salary'])
sns.violinplot(df['Salary'])
# plt.title("Boxlpot of 'Salary'")
plt.title("Violin plot of 'Salary'")
```

```
/Users/Code/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:3
6: FutureWarning: Pass the following variable as a keyword arg: x. From versio
n 0.12, the only valid positional argument will be `data`, and passing other a
rguments without an explicit keyword will result in an error or misinterpretat
ion.
  warnings.warn(
/Users/Code/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:3
6: FutureWarning: Pass the following variable as a keyword arg: x. From versio
n 0.12, the only valid positional argument will be `data`, and passing other a
rguments without an explicit keyword will result in an error or misinterpretat
ion.
  warnings.warn(
```

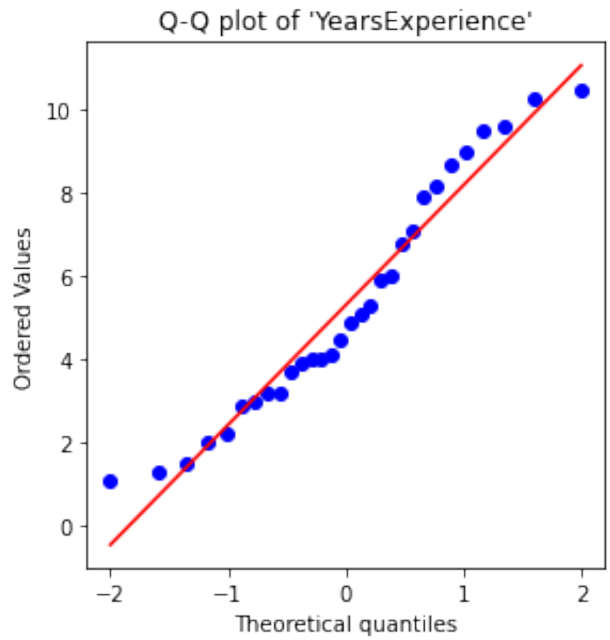
Out[11]:
```
Text(0.5, 1.0, "Violin plot of 'Salary'")
```

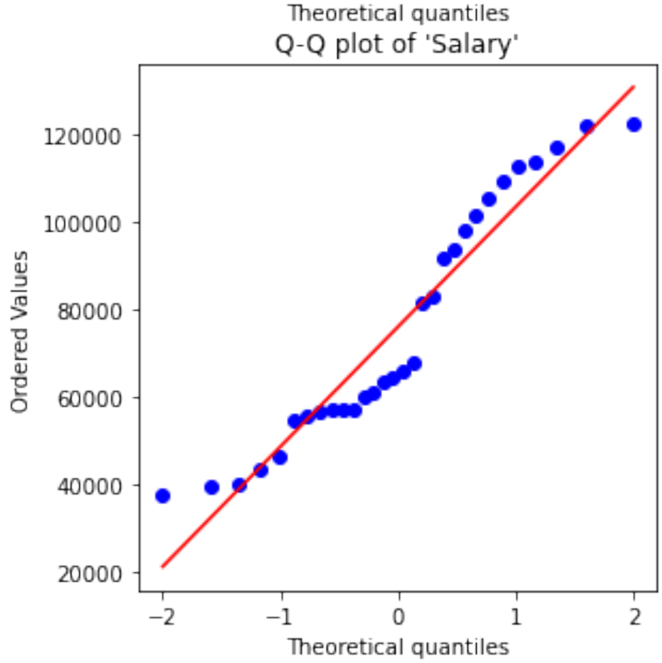




In [13]:
```
# Normal Q-Q plot
plt.figure(figsize=(20,10))
plt.subplot(2,4,4)
probplot(df['YearsExperience'], plot=plt)
plt.title("Q-Q plot of 'YearsExperience'")
plt.subplot(2,4,8)
probplot(df['Salary'], plot=plt)
plt.title("Q-Q plot of 'Salary'")
```

Out[13]:
```
Text(0.5, 1.0, "Q-Q plot of 'Salary'")
```





From the above graphical representations, we can say there are no outliers in our data, and YearsExperience looks like normally distributed, and Salary doesn't look normal. We can verify this using Shapiro Test.

Check if there is any correlation between the variables using df.corr()

In [16]:

```
print("Correlation: "+ 'n', df.corr()) # 0.978 which is high positive correla
# Draw a heatmap for correlation matrix
plt.subplot(1,1,1)
sns.heatmap(df.corr(), annot=True)
```

```
Correlation: n                  YearsExperience    Salary
YearsExperience          1.000000   0.978242
Salary                   0.978242   1.000000
<AxesSubplot:>
```

Out[16]:

Linear Regression using scikit-learn

LinearRegression(): LinearRegression fits a linear model with coefficients β = (β1, ..., βp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

In [19]:

```
#     defining the independent and dependent features
x= df.iloc[:, 1:2]
y= df.iloc[:, 0:1]
    # print(x,y)
```

In [21]:
```
# Instantiating the LinearRegression object
regressor = LinearRegression()
```

In [39]:
```
model = smf.ols('Salary ~ YearsExperience', data = df)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Salary   R-squared:                       0.957
Model:                            OLS   Adj. R-squared:                  0.955
Method:                 Least Squares   F-statistic:                     622.5
Date:                Fri, 15 Apr 2022   Prob (F-statistic):           1.14e-20
Time:                        16:19:17   Log-Likelihood:                -301.44
No. Observations:                  30   AIC:                             606.9
Df Residuals:                      28   BIC:                             609.7
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
=====
                      coef    std err          t      P>|t|      [0.025      0
.975]
------------------------------------------------------------------------------
-----
Intercept        2.579e+04   2273.053     11.347      0.000    2.11e+04    3.0
4e+04
YearsExperience  9449.9623    378.755     24.950      0.000    8674.119    1.0
2e+04
==============================================================================
Omnibus:                        2.140   Durbin-Watson:                   1.648
Prob(Omnibus):                  0.343   Jarque-Bera (JB):                1.569
Skew:                           0.363   Prob(JB):                        0.456
Kurtosis:                       2.147   Cond. No.                         13.2
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
```

In [22]:
```
# Training the model
regressor.fit(x,y)
```

Out[22]: LinearRegression()

In [23]:
```
# Checking the coefficients for the prediction of each of the predictor
print('n'+"Coeff of the predictor: ",regressor.coef_)
```

```
nCoeff of the predictor:  [[0.00010127]]
```

In [24]:

```
# Checking the intercept
print("Intercept: ",regressor.intercept_)
```

```
Intercept:  [-2.38316056]
```

In [28]:

```
# Predicting the output
y_pred = regressor.predict(x)
#print(y_pred)
```

In [27]:

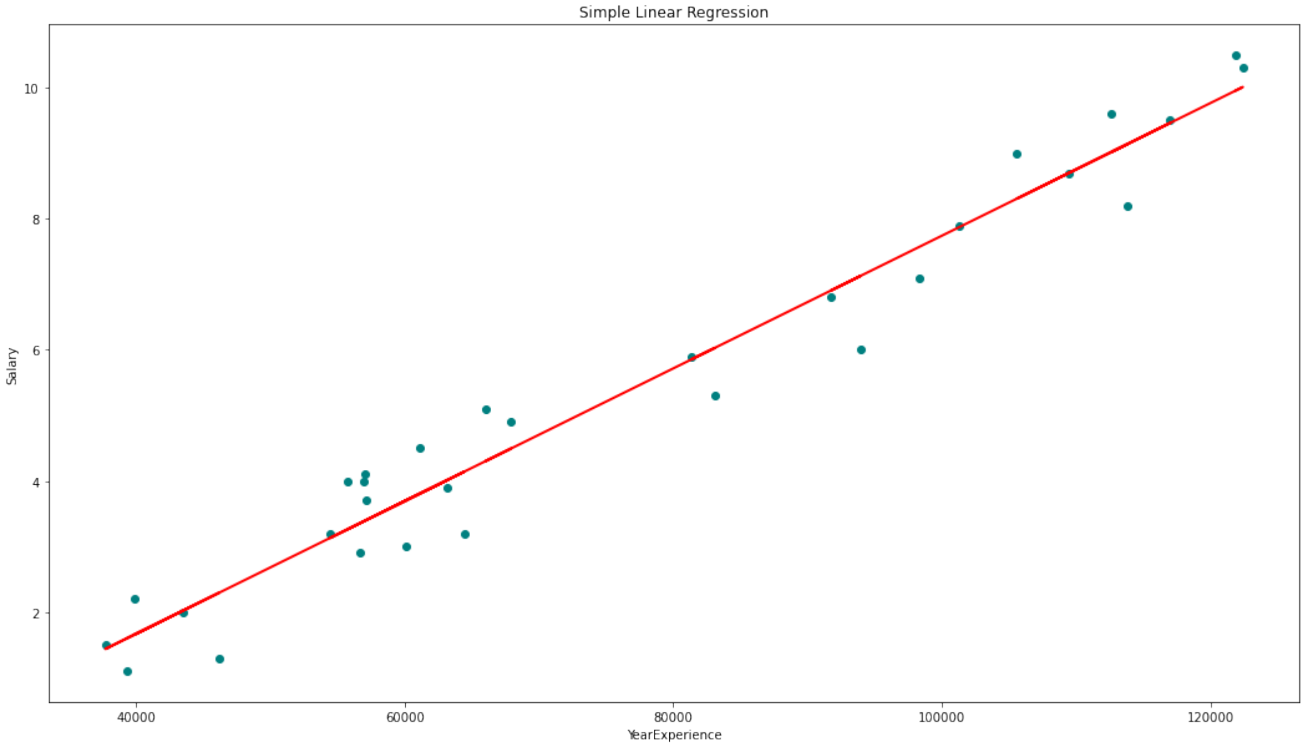
```
# Checking the MSE
print("Mean squared error(MSE): %.2f" % mean_squared_error(y, y_pred))
# Checking the R2 value
print("Coefficient of determination: %.3f" % r2_score(y, y_pred)) # Evaluates
```

```
Mean squared error(MSE): 0.34
Coefficient of determination: 0.957
```

In [29]:

```
# visualizing the results.
plt.figure(figsize=(18, 10))
# Scatter plot of input and output values
plt.scatter(x, y, color='teal')
# plot of the input and predicted output values
plt.plot(x, regressor.predict(x), color='Red', linewidth=2 )
plt.title('Simple Linear Regression')
plt.xlabel('YearExperience')
plt.ylabel('Salary')
```

Out[29]: Text(0, 0.5, 'Salary')



In [ ]:

In [ ]:

Multiple Regression Model

We can create a regression model using more than one explanatory variables. Let's load a dataset and do it.

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from scipy import stats
from scipy.stats import probplot
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn import preprocessing
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

In [3]:
```
df = pd.read_csv("HeartDiseaseTrain.csv")
df.head(n=6)
```

Out[3]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	targe
0	63	1	1	145	233	1	2	150	0	2.3	3	2	3	(
1	67	1	4	160	286	0	2	108	1	1.5	2	5	2	'
2	67	1	4	120	229	0	2	129	1	2.6	2	4	4	'
3	37	1	3	130	250	0	0	187	0	3.5	3	2	2	(
4	41	0	2	130	204	0	2	172	0	1.4	1	2	2	(
5	56	1	2	120	236	0	0	178	0	0.8	1	2	2	(

Understanding the dataset using Data analysis

In [4]:
```
#checking the data types of the columns
df.dtypes
```

Out[4]:
```
age            int64
sex            int64
cp             int64
trestbps       int64
chol           int64
fbs            int64
restecg        int64
thalach        int64
exang          int64
oldpeak      float64
slope          int64
ca             int64
thal           int64
target         int64
dtype: object
```

In [5]:
```
#checking if there is a missing row or observation in the dataset
df.isnull().values.any()
```

Out[5]:
```
False
```

In [6]:
```
#checking the number of missing values for each of the column
df.isnull().sum()
```

Out[6]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

In [7]:
```
print(df.describe(include='all'))
```

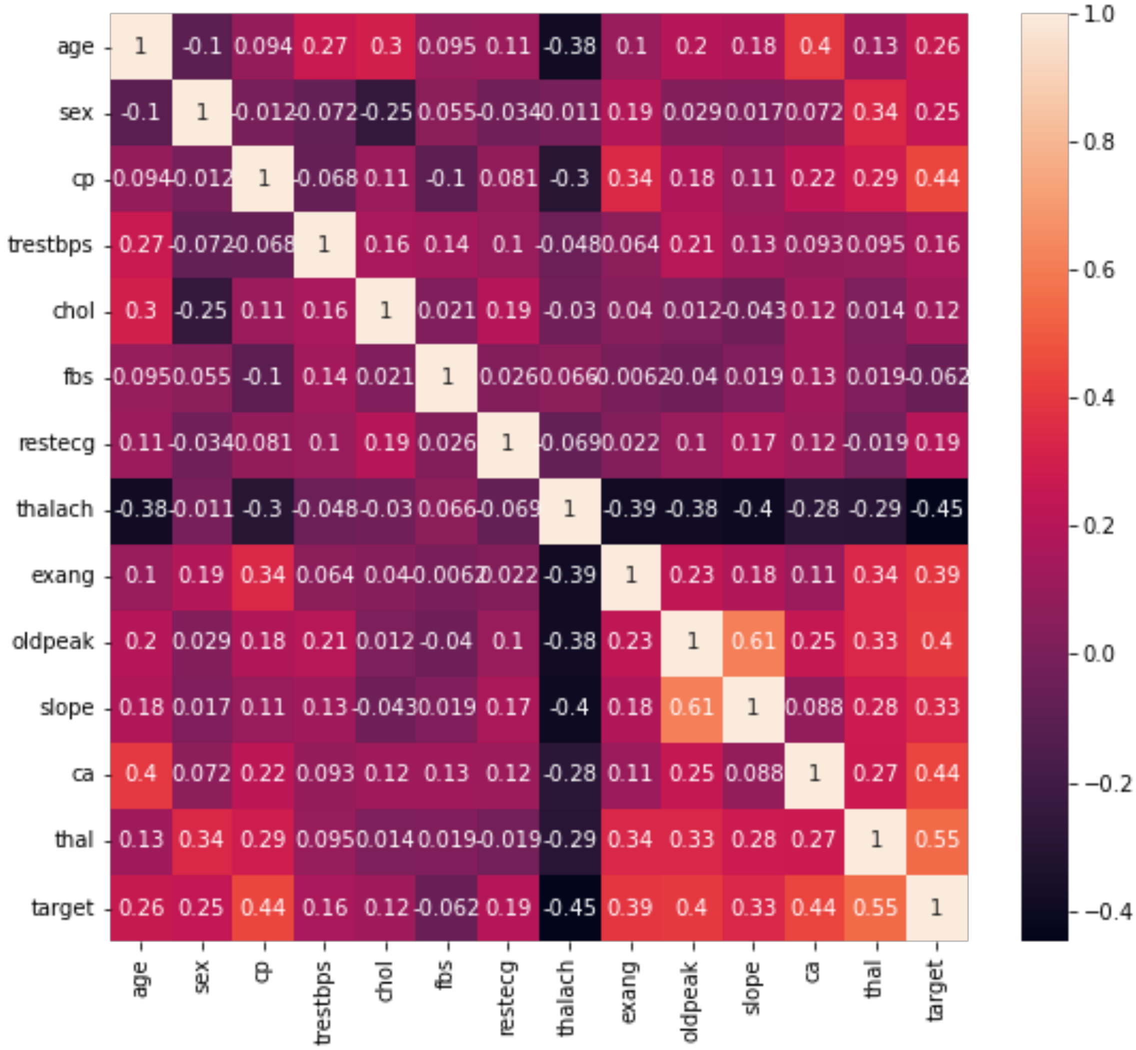
```
                age         sex          cp    trestbps        chol         fbs  \
count  200.000000  200.000000  200.000000  200.000000  200.000000  200.000000
mean    54.745000    0.710000    3.155000  132.565000  252.655000    0.170000
std      8.800981    0.454901    0.956845   18.025269   54.316086    0.376575
min     29.000000    0.000000    1.000000   94.000000  126.000000    0.000000
25%     48.750000    0.000000    3.000000  120.000000  218.500000    0.000000
50%     56.000000    1.000000    3.000000  130.000000  248.000000    0.000000
75%     61.000000    1.000000    4.000000  140.000000  282.250000    0.000000
max     77.000000    1.000000    4.000000  200.000000  564.000000    1.000000

           restecg     thalach       exang     oldpeak       slope          ca  \
count  200.000000  200.000000  200.000000  200.000000  200.000000  200.000000
mean     1.120000  151.105000    0.330000    1.116000    1.620000    2.695000
std      0.995265   22.244506    0.471393    1.171669    0.638465    0.972989
min      0.000000   88.000000    0.000000    0.000000    1.000000    2.000000
25%      0.000000  139.000000    0.000000    0.000000    1.000000    2.000000
50%      2.000000  154.500000    0.000000    0.800000    2.000000    2.000000
75%      2.000000  166.000000    1.000000    1.650000    2.000000    3.000000
max      2.000000  202.000000    1.000000    6.200000    3.000000    5.000000

              thal      target
count  200.000000  200.000000
mean     2.900000    0.450000
std      0.971969    0.498742
min      2.000000    0.000000
25%      2.000000    0.000000
50%      2.000000    0.000000
75%      4.000000    1.000000
max      4.000000    1.000000
```

We can also check the correlation between the variables using a correlation plot

In [8]:

```
# Draw a heatmap for correlation matrix
plt.figure(figsize=(9,8))
plt.subplot(1,1,1)
sns.heatmap(df.corr(), annot=True)
```

Out[8]: <AxesSubplot:>



Now let's create a model taking chol as our response variable and age and trestbps as our explanatory variable.

In [9]:

```
model = smf.ols('chol~trestbps+age', data = df)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   chol   R-squared:                       0.097
Model:                            OLS   Adj. R-squared:                  0.088
Method:                 Least Squares   F-statistic:                     10.64
Date:                Mon, 25 Apr 2022   Prob (F-statistic):           4.10e-05
Time:                        15:49:10   Log-Likelihood:                -1072.0
No. Observations:                 200   AIC:                             2150.
Df Residuals:                     197   BIC:                             2160.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    125.4077     31.767      3.948      0.000      62.760     188.055
trestbps       0.2446      0.212      1.156      0.249      -0.173       0.662
age            1.7322      0.433      3.998      0.000       0.878       2.587
==============================================================================
Omnibus:                       64.293   Durbin-Watson:                   2.125
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              256.300
Skew:                           1.212   Prob(JB):                     2.21e-56
Kurtosis:                       7.988   Cond. No.                     1.25e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The condition number is large, 1.25e+03. This might indicate that there ar
e
strong multicollinearity or other numerical problems.
```

Now let's ass the variable thalach in the model.

In [10]:

```
model = smf.ols('chol~trestbps+age+thal', data = df)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   chol   R-squared:                       0.098
Model:                            OLS   Adj. R-squared:                  0.085
Method:                 Least Squares   F-statistic:                     7.126
Date:                Mon, 25 Apr 2022   Prob (F-statistic):           0.000144
Time:                        15:49:10   Log-Likelihood:                -1071.9
No. Observations:                 200   AIC:                             2152.
Df Residuals:                     196   BIC:                             2165.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    128.3651     32.542      3.945      0.000      64.188     192.543
trestbps       0.2505      0.212      1.179      0.240      -0.168       0.669
age            1.7525      0.437      4.013      0.000       0.891       2.614
thal          -1.6756      3.829     -0.438      0.662      -9.227       5.876
==============================================================================
Omnibus:                       65.751   Durbin-Watson:                   2.125
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              268.767
Skew:                           1.234   Prob(JB):                     4.35e-59
Kurtosis:                       8.115   Cond. No.                     1.28e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The condition number is large, 1.28e+03. This might indicate that there ar
e
strong multicollinearity or other numerical problems.
```

Adding Interaction term

we can add interaction term in two ways one is interaction term and the original term using * and the other is using :: with just the interaction term.

In [11]:

```
model = smf.ols('chol~trestbps*age', data = df)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   chol   R-squared:                       0.109
Model:                            OLS   Adj. R-squared:                  0.096
Method:                 Least Squares   F-statistic:                     8.015
Date:                Mon, 25 Apr 2022   Prob (F-statistic):           4.58e-05
Time:                        15:49:10   Log-Likelihood:                -1070.7
No. Observations:                 200   AIC:                             2149.
Df Residuals:                     196   BIC:                             2163.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
==
                   coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
Intercept     -194.1734    200.754     -0.967      0.335    -590.089     201.7
42
trestbps         2.7245      1.553      1.755      0.081      -0.338       5.7
87
age              7.3871      3.534      2.090      0.038       0.417      14.3
57
trestbps:age    -0.0437      0.027     -1.612      0.109      -0.097       0.0
10
==============================================================================
Omnibus:                       61.076   Durbin-Watson:                   2.122
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              223.726
Skew:                           1.176   Prob(JB):                     2.62e-49
Kurtosis:                       7.616   Cond. No.                     4.12e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The condition number is large, 4.12e+05. This might indicate that there ar
e
strong multicollinearity or other numerical problems.
```

In [12]:

```
model = smf.ols('chol~trestbps:age', data = df)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   chol   R-squared:                       0.080
Model:                            OLS   Adj. R-squared:                  0.076
Method:                 Least Squares   F-statistic:                     17.26
Date:                Mon, 25 Apr 2022   Prob (F-statistic):           4.85e-05
Time:                        15:49:10   Log-Likelihood:                -1073.9
No. Observations:                 200   AIC:                             2152.
Df Residuals:                     198   BIC:                             2158.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
==
                  coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
Intercept     187.6823     16.070     11.679      0.000     155.991     219.3
73
trestbps:age    0.0089      0.002      4.154      0.000       0.005       0.0
13
==============================================================================
Omnibus:                       71.861   Durbin-Watson:                   2.135
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              332.678
Skew:                           1.316   Prob(JB):                     5.75e-73
Kurtosis:                       8.744   Cond. No.                     3.26e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The condition number is large, 3.26e+04. This might indicate that there ar
e
strong multicollinearity or other numerical problems.
```

Categorical data

Categorical data is very useful in data science. We will see now how to manipulate categorical data and also how to use categorical data to build regression model.

We can make a column of a dataframe from numerical to categorical if feasible.

In [13]:

```
df['sex']=df['sex'].astype('category')
##we can rename the column values as male and female
df['sex'].replace({1:"M",0:"F"},inplace=True)
df['sex']=df['sex'].astype('category')
print(df.describe(include='category'))
```

```
        sex
count   200
unique    2
top       M
freq    142
```

In [14]:
```
df.head(n=5)
```

Out[14]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	targe
0	63	M	1	145	233	1	2	150	0	2.3	3	2	3	(
1	67	M	4	160	286	0	2	108	1	1.5	2	5	2	'
2	67	M	4	120	229	0	2	129	1	2.6	2	4	4	'
3	37	M	3	130	250	0	0	187	0	3.5	3	2	2	(
4	41	F	2	130	204	0	2	172	0	1.4	1	2	2	(

Now we can see that the sex is now categorical as M and F

In [15]:
```
df.dtypes
```

Out[15]:
```
age              int64
sex           category
cp               int64
trestbps         int64
chol             int64
fbs              int64
restecg          int64
thalach          int64
exang            int64
oldpeak        float64
slope            int64
ca               int64
thal             int64
target           int64
dtype: object
```

Building model with categorical data

In [16]:
```
model = smf.ols('chol~sex', data = df)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   chol   R-squared:                       0.064
Model:                            OLS   Adj. R-squared:                  0.060
Method:                 Least Squares   F-statistic:                     13.63
Date:                Mon, 25 Apr 2022   Prob (F-statistic):           0.000287
Time:                        15:49:10   Log-Likelihood:                -1075.6
No. Observations:                 200   AIC:                             2155.
Df Residuals:                     198   BIC:                             2162.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    274.1724      6.916     39.644      0.000     260.534     287.811
sex[T.M]     -30.3062      8.208     -3.692      0.000     -46.492     -14.121
==============================================================================
Omnibus:                       49.018   Durbin-Watson:                   2.161
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              165.853
Skew:                           0.949   Prob(JB):                     9.67e-37
Kurtosis:                       7.038   Cond. No.                         3.48
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
```

In the summary table the sex[T.M] means that the type specified for sex is Male. T goes for type.

In [17]:

```
#Adding numerical variable with categorical
model = smf.ols('chol~trestbps+sex', data = df)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   chol   R-squared:                       0.083
Model:                            OLS   Adj. R-squared:                  0.074
Method:                 Least Squares   F-statistic:                     8.961
Date:                Mon, 25 Apr 2022   Prob (F-statistic):           0.000188
Time:                        15:49:10   Log-Likelihood:                -1073.5
No. Observations:                 200   AIC:                             2153.
Df Residuals:                     197   BIC:                             2163.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    218.1654     28.576      7.635      0.000     161.811     274.520
sex[T.M]     -29.1116      8.166     -3.565      0.000     -45.216     -13.007
trestbps       0.4161      0.206      2.019      0.045       0.010       0.823
==============================================================================
Omnibus:                       56.894   Durbin-Watson:                   2.148
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              228.726
Skew:                           1.051   Prob(JB):                     2.15e-50
Kurtosis:                       7.799   Cond. No.                     1.04e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The condition number is large, 1.04e+03. This might indicate that there ar
e
strong multicollinearity or other numerical problems.
```

In [18]:

```
#adding interaction term with categorical data
model = smf.ols('chol~trestbps*sex', data = df)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   chol   R-squared:                       0.083
Model:                            OLS   Adj. R-squared:                  0.069
Method:                 Least Squares   F-statistic:                     5.946
Date:                Mon, 25 Apr 2022   Prob (F-statistic):           0.000669
Time:                        15:49:10   Log-Likelihood:                -1073.5
No. Observations:                 200   AIC:                             2155.
Df Residuals:                     196   BIC:                             2168.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
=======
                        coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-------
Intercept           214.8519     48.513      4.429      0.000     119.178
310.526
sex[T.M]            -24.1618     59.054     -0.409      0.683    -140.624
92.301
trestbps              0.4407      0.357      1.235      0.218      -0.263
1.144
trestbps:sex[T.M]    -0.0370      0.438     -0.085      0.933      -0.900
0.826
==============================================================================
Omnibus:                       57.224   Durbin-Watson:                   2.145
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              231.410
Skew:                           1.056   Prob(JB):                     5.62e-51
Kurtosis:                       7.828   Cond. No.                     3.30e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The condition number is large, 3.3e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

We should remember that if we use the categorical columns as numerical like 0 and 1 we will get the same results. We can also use C() in the model to factor them inside the model.

In [19]:

```
model = smf.ols('chol~trestbps+C(cp)', data = df)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   chol   R-squared:                       0.039
Model:                            OLS   Adj. R-squared:                  0.019
Method:                 Least Squares   F-statistic:                     1.978
Date:                Mon, 25 Apr 2022   Prob (F-statistic):             0.0994
Time:                        15:49:10   Log-Likelihood:                -1078.3
No. Observations:                 200   AIC:                             2167.
Df Residuals:                     195   BIC:                             2183.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    168.8236     33.850      4.987      0.000     102.065     235.582
C(cp)[T.2]     9.8141     16.945      0.579      0.563     -23.605      43.233
C(cp)[T.3]    17.1428     15.375      1.115      0.266     -13.180      47.466
C(cp)[T.4]    22.2369     14.750      1.508      0.133      -6.853      51.327
trestbps       0.5038      0.216      2.334      0.021       0.078       0.930
==============================================================================
Omnibus:                       75.125   Durbin-Watson:                   2.170
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              372.089
Skew:                           1.360   Prob(JB):                     1.59e-81
Kurtosis:                       9.104   Cond. No.                     1.32e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The condition number is large, 1.32e+03. This might indicate that there ar
e
strong multicollinearity or other numerical problems.
```

In [20]:

```
##we can predict for a new value
preds = results.predict(pd.DataFrame({"trestbps":[100],"cp":[1]}))
print(preds)
```

```
0    219.206839
dtype: float64
```

In [21]:

```
#predicting for train
preds1 = results.predict(df)

df["predicted"] = preds1
```

In [22]:

```
df.head()
```

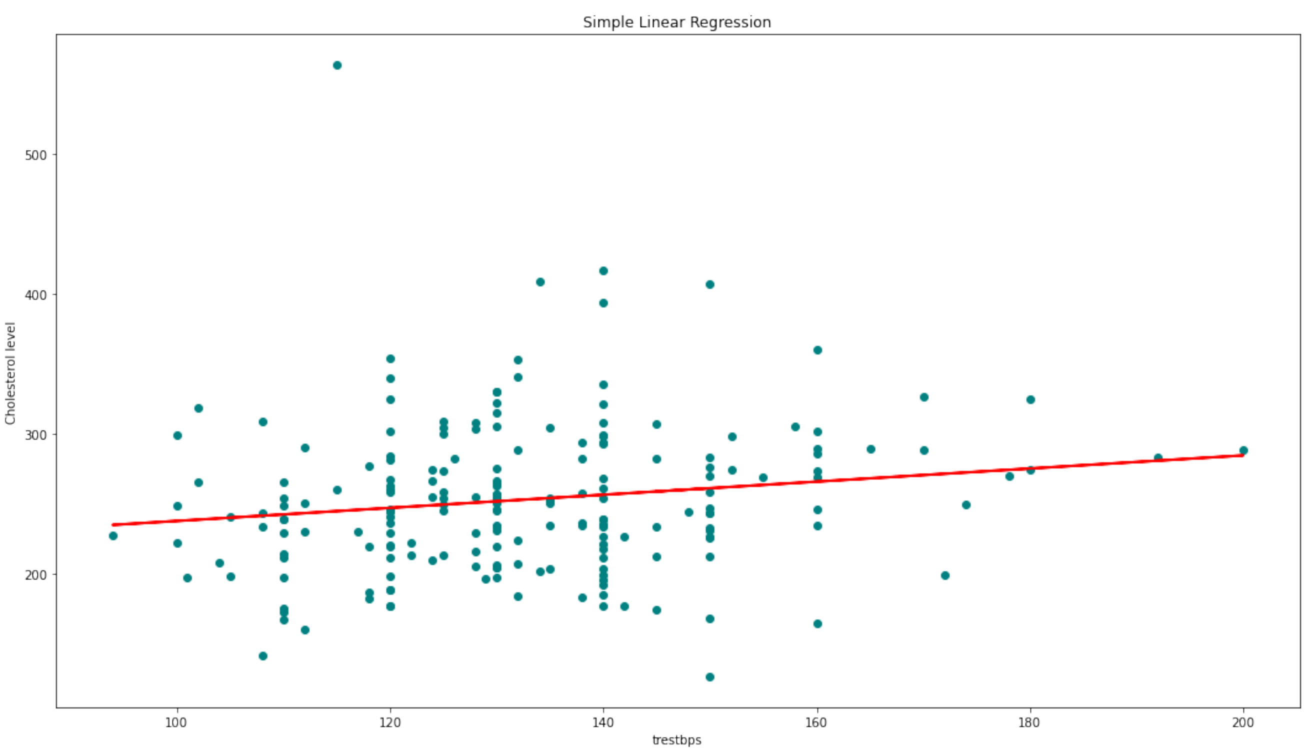
Out[22]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	targe
0	63	M	1	145	233	1	2	150	0	2.3	3	2	3	(
1	67	M	4	160	286	0	2	108	1	1.5	2	5	2	'
2	67	M	4	120	229	0	2	129	1	2.6	2	4	4	'
3	37	M	3	130	250	0	0	187	0	3.5	3	2	2	(
4	41	F	2	130	204	0	2	172	0	1.4	1	2	2	(

In [23]:

```
model = smf.ols('chol~trestbps', data = df)
results = model.fit()
preds = results.predict()
# visualizing the results.
plt.figure(figsize=(18, 10))
# Scatter plot of input and output values
plt.scatter(df.trestbps, df.chol, color='teal')
# plot of the input and predicted output values
plt.plot(df.trestbps, results.predict(), color='Red', linewidth=2 )
plt.title('Simple Linear Regression')
plt.xlabel('trestbps')
plt.ylabel('Cholesterol level')
```

Out[23]: Text(0, 0.5, 'Cholesterol level')



In [ ]: