Types of Regression & Classification

1. Ridge Regression
2. Lasso Regression
3. ElasticNet Regression
4. Logistic Regression

Lasso, Ridge and ElasticNet Regression

import libs

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import seaborn as sb
sb.set(style='whitegrid')
```

import dataset

```
# from google.colab import files
# uploaded = files.upload()
# D7data1.csv
# Boston Housing Dataset

import os
os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
os.getcwd()
```

```
'C:\\Users\\surya\\Downloads\\PG-DBDA-Mar23\\Datasets'
```

```
dataset = pd.read_csv('D7data1.csv')
# Boston Housing Dataset
dataset.head()
```

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	me
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	2
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	2
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	3
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	3
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	3

```
1 dataset.shape
```

```
(506, 14)
```

```
1 dataset.describe()
```

	crim	zn	indus	chas	nox	rm	age	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	50
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	1.

```
1 dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   crim     506 non-null    float64
 1   zn       506 non-null    float64
 2   indus    506 non-null    float64
 3   chas     506 non-null    int64  
 4   nox      506 non-null    float64
 5   rm       506 non-null    float64
 6   age      506 non-null    float64
```

```
 7   dis      506 non-null    float64
 8   rad      506 non-null    int64
 9   tax      506 non-null    int64
 10  ptratio  506 non-null    float64
 11  b        506 non-null    float64
 12  lstat    506 non-null    float64
 13  medv     506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

EDA

null check

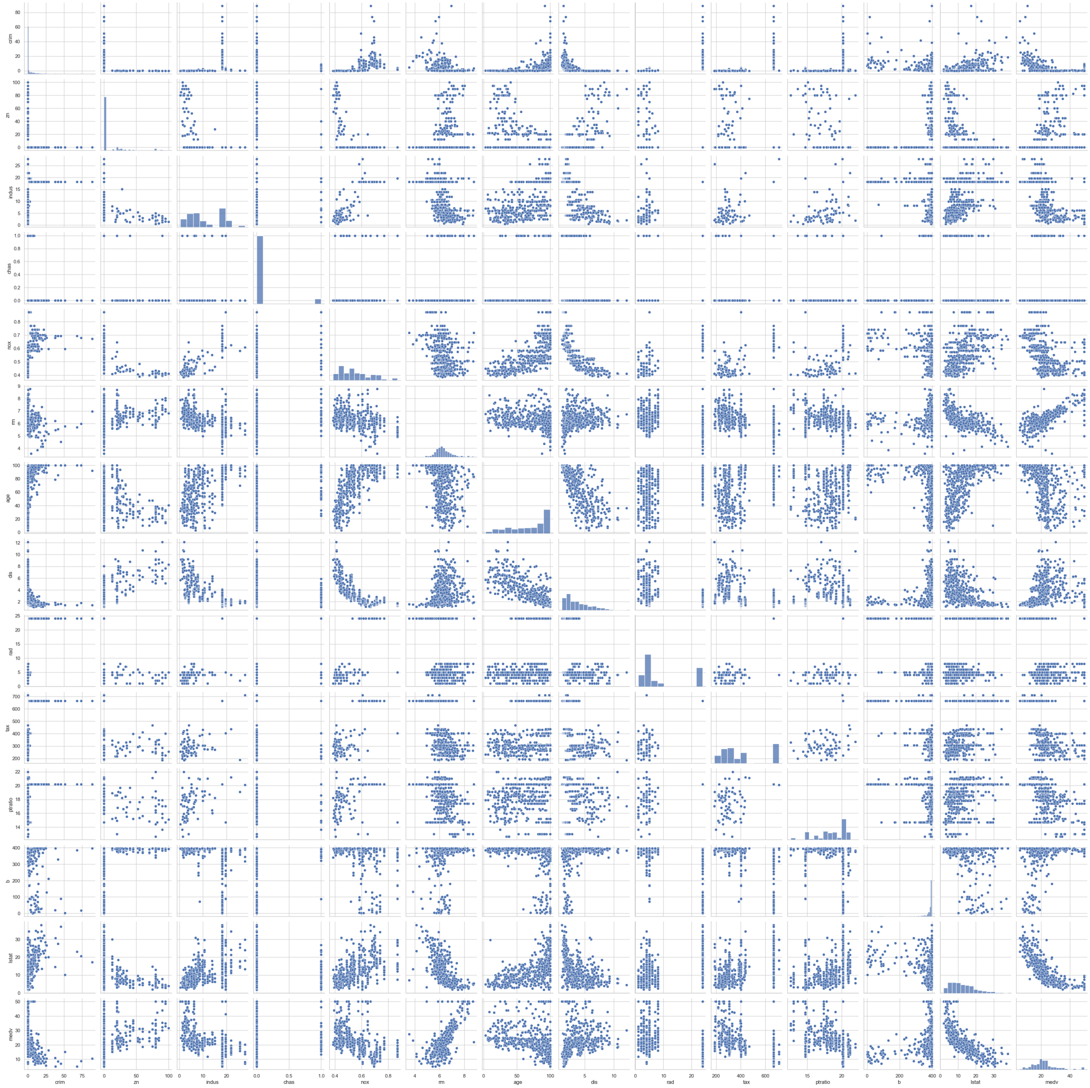
```
dataset.isnull().sum()
```

```
crim       0
zn         0
indus      0
chas       0
nox        0
rm         0
age        0
dis        0
rad        0
tax        0
ptratio    0
b          0
lstat      0
medv       0
dtype: int64
```

```
import seaborn as sns
```

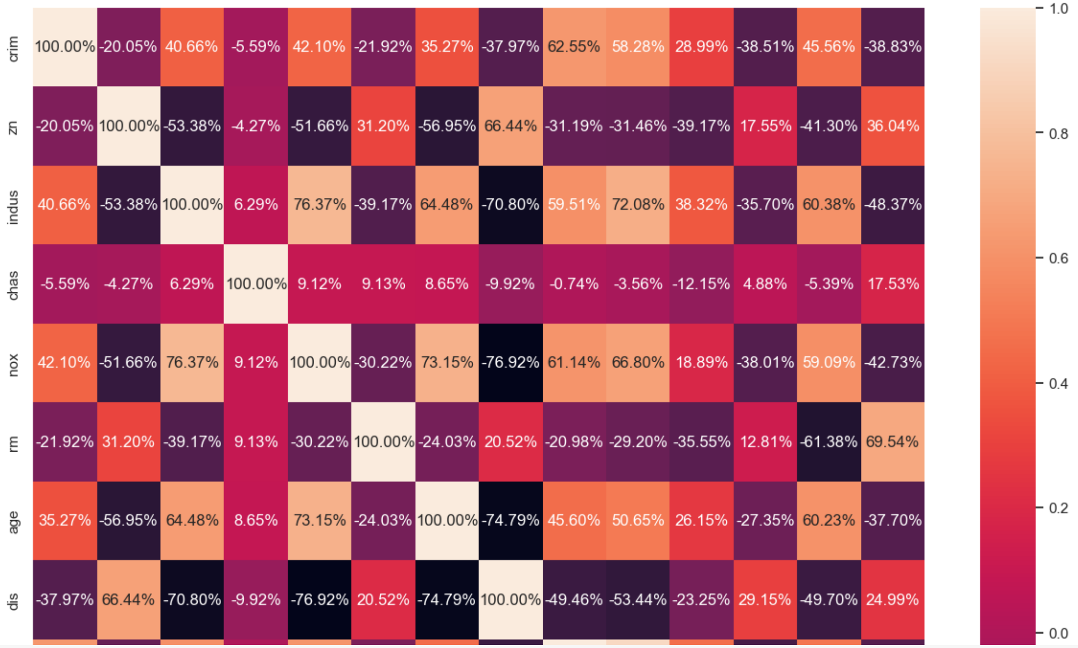
```
sns.pairplot(dataset)
```

```
c:\users\surya\appdata\local\programs\python\python39\lib\site-packages\seaborn\axisgrid.p
  self._figure.tight_layout(*args, **kwargs)
<seaborn.axisgrid.PairGrid at 0x1fab93ea970>
```



```
plt.figure(figsize=(9, 9))
sb.heatmap(dataset.corr(), annot=True)
plt.show()
```


```
plt.figure(figsize=(15,15))
sb.heatmap(dataset.corr(), annot=True, fmt='.2%')
plt.show()
```




```
# To-Do : Visualize correlation of features
corr_medv = dataset.corrwith(dataset['medv'])
# prints correlation of one column with other columns
corr_medv
```

```
crim      -0.388305
zn         0.360445
indus     -0.483725
chas       0.175260
nox       -0.427321
rm         0.695360
age       -0.376955
dis        0.249929
rad       -0.381626
tax       -0.468536
ptratio   -0.507787
b          0.333461
lstat     -0.737663
medv       1.000000
dtype: float64
```

```
corr_medv.abs().sort_values(ascending=False)
# sorting according to magnitudes
```

```
medv       1.000000
lstat      0.737663
```

```
rm         0.695360
ptratio    0.507787
indus      0.483725
tax        0.468536
nox        0.427321
crim       0.388305
rad        0.381626
age        0.376955
zn         0.360445
b          0.333461
dis        0.249929
chas       0.175260
dtype: float64
```

```
dataset.corr()
# prints correlation of all columns with other columns
```

	crim	zn	indus	chas	nox	rm	age	dis	
crim	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670	0
zn	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408	-0
indus	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027	0
chas	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176	-0
nox	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230	0
rm	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246	-0
age	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881	0
dis	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000	-0
rad	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588	1
tax	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432	0
ptratio	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471	0
b	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512	-0
lstat	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996	0
medv	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.249929	-0

identify X & Y

```
# independent vars
x = dataset.iloc[ : , :13].values
x[:2]
```

```
array([[6.3200e-03, 1.8000e+01, 2.3100e+00, 0.0000e+00, 5.3800e-01,
        6.5750e+00, 6.5200e+01, 4.0900e+00, 1.0000e+00, 2.9600e+02,
        1.5300e+01, 3.9690e+02, 4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, 0.0000e+00, 4.6900e-01,
        6.4210e+00, 7.8900e+01, 4.9671e+00, 2.0000e+00, 2.4200e+02,
        1.7800e+01, 3.9690e+02, 9.1400e+00]])
```

```
# dependent vars
y = dataset.iloc[ : , 13].values
y[:2]
```

```
array([24. , 21.6])
```

Splitting

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

Preprocessing

Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)
```

```
x_train[:2]
```

```
array([[-0.37257438, -0.49960763, -0.70492455,  3.66450153, -0.42487874,
         0.93567804,  0.69366877, -0.4372179 , -0.16224243, -0.56165616,
        -0.48463784,  0.3716906 , -0.41100022],
       [-0.39709866, -0.49960763, -0.04487755, -0.27288841, -1.24185891,
```

```
       -0.49118121, -1.8355285 ,  0.73005474, -0.62464765, -0.57337637,
        0.33649132,  0.20501196, -0.38768057]])
```

```
x_test[:2]
```

```
array([[-4.36752612e-01, -4.39882694e-01, -1.26009787e+00,
        -2.71448357e-01, -7.59976330e-01,  1.59563423e-01,
        -1.75561320e+00,  6.00755701e-01, -6.91310354e-01,
         2.55645541e-03, -7.48346275e-01,  2.67413362e-01,
        -7.88042853e-01],
       [ 4.63542006e-01, -4.39882694e-01,  1.09537107e+00,
        -2.71448357e-01,  6.84309089e-01, -9.00880536e-04,
         1.17611331e+00, -1.23857700e+00,  1.56512664e+00,
         1.50350755e+00,  8.67390455e-01,  1.77343807e-01,
        -4.49823527e-01]])
```

Linear Regression

Modeling: Linear Regression

```
from sklearn.linear_model import LinearRegression
```

```
lm = LinearRegression()
```

Training: Linear Regression

```
lm.fit(x_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

calculating Linear coefficients

```
lm_coeff = pd.Series(lm.coef_, index=dataset.columns[: 13])
# storing coeffients of Linear Regression model as a Pandas Series
lm_coeff
```

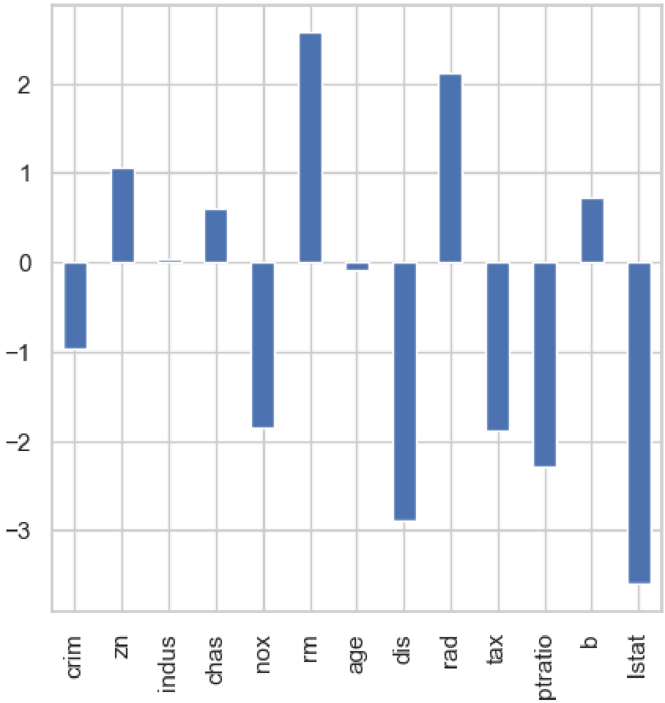
```
crim      -0.970820
zn         1.057149
```

```
indus      0.038311
chas       0.594506
nox       -1.855148
rm         2.573219
age       -0.087615
dis       -2.880943
rad        2.112245
tax       -1.875331
ptratio   -2.292767
b          0.718179
lstat     -3.592455
dtype: float64
```

visualizing: Linear coefficients

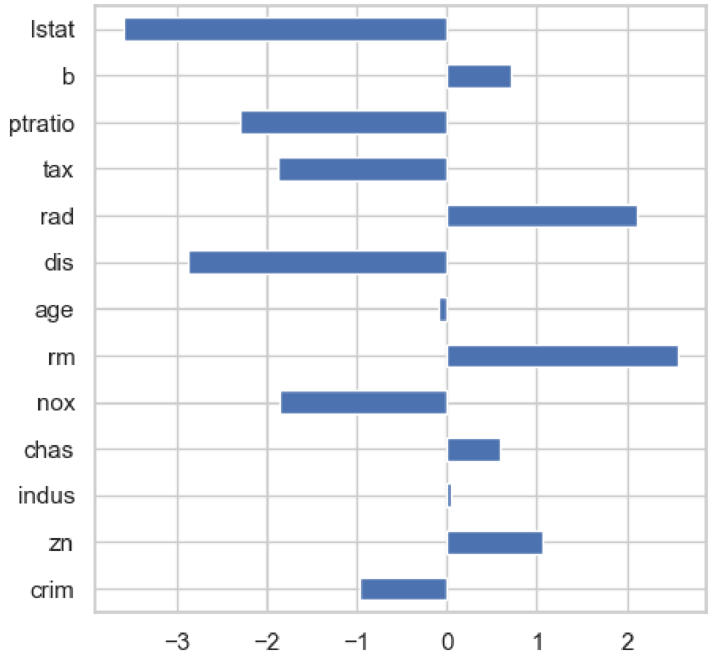
```
plt.figure(figsize=(5, 5))
lm_coeff.plot(kind='bar')
# plotting bar graph
```

```
<Axes: >
```



```
plt.figure(figsize=(5, 5))
lm_coeff.plot(kind='barh')
# plotting horizontal bar graph
```

<Axes: >



Ridge Regression

Modeling: Ridge Regression

```
from sklearn.linear_model import Ridge
```

```
rid = Ridge()
```

Training: Ridge Regression

```
rid.fit(x_train, y_train)
```

```
▾ Ridge
Ridge()
```

Calculating Coefficients: Ridge Regression

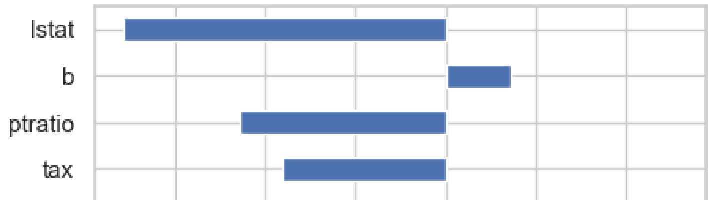
```
rd_coeff = pd.Series(rid.coef_, index=dataset.columns[: 13])
# storing coeffients of Linear Regression model as a Pandas Series
rd_coeff
```

```
crim      -0.962257
zn         1.040872
indus      0.011680
chas       0.598719
nox       -1.820134
rm         2.583786
age       -0.095188
dis       -2.848263
rad        2.036231
tax       -1.806092
ptratio   -2.283191
b          0.718310
lstat     -3.576073
dtype: float64
```

Visualizing Coefficients: Ridge Regression

```
plt.figure(figsize=(5, 5))
rd_coeff.plot(kind='barh')
# plotting bar graph
```

<Axes: >



Lasso Regression

Modeling: Lasso Regression

```
from sklearn.linear_model import Lasso
```

```
la = Lasso()
```

Training: Lasso Regression

```
la.fit(x_train, y_train)
```

```
▾ Lasso
Lasso()
```

Calculating Coefficients: Lasso Regression

```
la_coeff = pd.Series(la.coef_, index=dataset.columns[: 13])
# storing coeffients of Linear Regression model as a Pandas Series
la_coeff
```

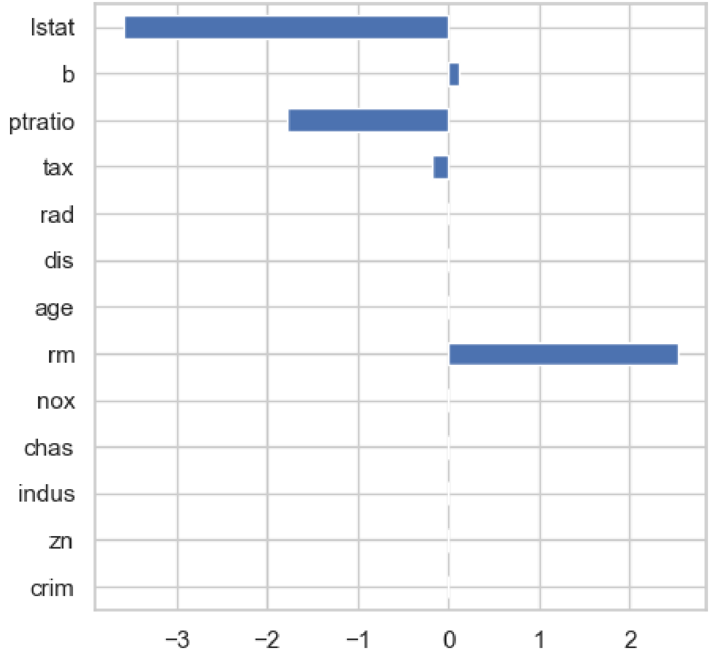
```
crim      -0.000000
zn         0.000000
indus     -0.000000
chas       0.000000
nox       -0.000000
rm         2.540098
age       -0.000000
dis       -0.000000
rad       -0.000000
tax       -0.171527
```

```
ptratio   -1.784796
b          0.110959
lstat     -3.585324
dtype: float64
```

Visualizing Coefficients: Lasso Regression

```
plt.figure(figsize=(5, 5))
la_coeff.plot(kind='barh')
# plotting bar graph
```

```
&lt;Axes: &gt;
```



ElasticNet Regression

Modeling: ElasticNet Regression

```
from sklearn.linear_model import ElasticNet
```

```
elnet = ElasticNet()
```

Training: ElasticNet Regression

```
elnet.fit(x_train, y_train)
```

```
▾ ElasticNet
ElasticNet()
```

Calculating Coeffiecients: ElasticNet Regression

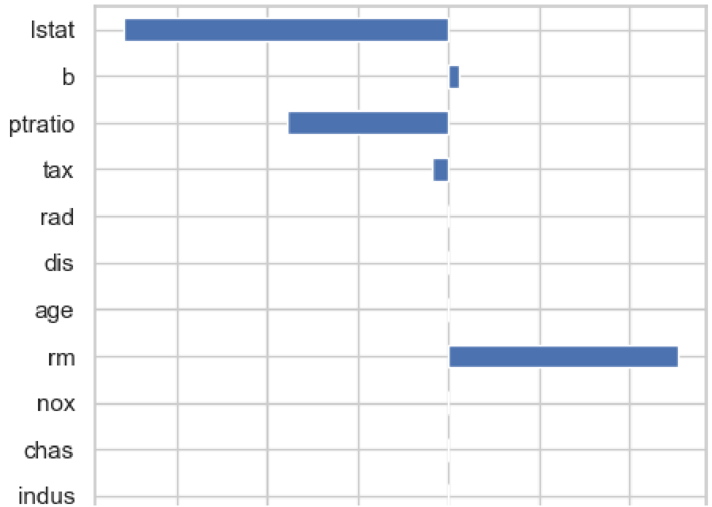
```
elnet_coeff = pd.Series(la.coef_, index=dataset.columns[: 13])
# storing coeffients of Linear Regression model as a Pandas Series
elnet_coeff
```

```
crim      -0.000000
zn         0.000000
indus     -0.000000
chas       0.000000
nox       -0.000000
rm         2.540098
age       -0.000000
dis       -0.000000
rad       -0.000000
tax       -0.171527
ptratio   -1.784796
b          0.110959
lstat     -3.585324
dtype: float64
```

Visualizing Coefficients: ElasticNet Regression

```
plt.figure(figsize=(5, 5))
elnet_coeff.plot(kind='barh')
# plotting bar graph
```

```
<Axes: >
```



Predictions

Prediction: Linear Regression

```
y_pred_lm = lm.predict(x_test)
y_pred_lm[:5]
```

```
array([25.87858248, 24.01180946, 30.22835216, 12.34741929, 21.99634736])
```

Prediction: Ridge Regression

```
y_pred_rid = rid.predict(x_test)
y_pred_rid[:5]
```

```
array([25.95510418, 23.95328489, 30.18542863, 12.34271115, 22.00978384])
```

Prediction: Lasso Regression

```
y_pred_la = la.predict(x_test)
y_pred_la[:5]
```

```
array([27.20745618, 22.43602729, 26.22245253, 13.77956784, 22.5613402 ])
```

Prediction: ElasticNet Regression

```
y_pred_elnet = elnet.predict(x_test)
y_pred_elnet[:5]
```

```
array([26.63669003, 20.77657954, 27.18734277, 13.83177378, 22.9013383 ])
```

Accuracy

```
from sklearn import metrics
```

Accuracy: Linear Regression

mean_squared_error

```
mse_lm = metrics.mean_squared_error(y_test, y_pred_lm)
mse_lm
# Mean Square Error
```

```
35.11642077929317
```

r2_score / coefficient of determination

```
metrics.r2_score(y_test, y_pred_lm)
# model accuracy using R-Square
# (coefficient of determination) regression score function.
```

```
0.5687450086990026
```

```
lm.score(x_test, y_test)
# accuracy for test dataset using linear model
# Return the coefficient of determination of the prediction
```

```
0.5687450086990026
```

Accuracy: Ridge Regression

mean_squared_error

```
mse_rid = metrics.mean_squared_error(y_test, y_pred_rid)
mse_rid
# Mean Square Error
```

```
35.18894572750182
```

r2_score / coefficient of determination

```
metrics.r2_score(y_test, y_pred_rid)
# model accuracy using R-Square
# (coefficient of determination) regression score function.
```

```
0.5678543499924846
```

```
rid.score(x_test, y_test)
# accuracy for test dataset
# Return the coefficient of determination of the prediction
```

```
0.5678543499924846
```

Accuracy: Lasso Regression

mean_squared_error

```
mse_la = metrics.mean_squared_error(y_test, y_pred_la)
mse_la
# Mean Square Error
```

```
40.62043710016924
```

r2_score / coefficient of determination

```
metrics.r2_score(y_test, y_pred_la)
# model accuracy using R-Square
```

```
# (coefficient of determination) regression score function.
```

```
0.5011517159343937
```

```
la.score(x_test, y_test)
# accuracy for test dataset
# Return the coefficient of determination of the prediction
```

```
0.5011517159343937
```

Accuracy: ElasicNet Regression

mean_squared_error

```
mse_elnet = metrics.mean_squared_error(y_test, y_pred_elnet)
mse_elnet
# Mean Square Error
```

```
43.25612264808443
```

r2_score / coefficient of determination

```
metrics.r2_score(y_test, y_pred_elnet)
# model accuracy using R-Square
# (coefficient of determination) regression score function.
```

```
0.46878359518592116
```

```
elnet.score(x_test, y_test)
# accuracy for test dataset
# Return the coefficient of determination of the prediction
```

```
0.46878359518592116
```

Ridge Penalty `α` value setting

```
from sklearn.model_selection import GridSearchCV
```

```
parameter = {'alpha':[1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 100]}
```

GridSearchCV

```
ridge_regressor = GridSearchCV(rid, parameter, cv=5)
# Exhaustive search over specified parameter values for an estimator
# GridSearchCV(estimator, param_grid, cv=None)
ridge_regressor.fit(x_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: Ridge
  ▸ Ridge
```

```
ridge_regressor.best_params_
# value of lambda
# Parameter setting that gave the best results on the hold out data.
```

```
{'alpha': 10}
```

```
ridge_regressor.best_score_
# if value of lambda is 10 then it gives best accuracy
# Mean cross-validated score of the best_estimator
```

```
0.7498253962279072
```

Logistic Regression

- used to classify into two classes

import libs

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

import dataset

```
# from google.colab import files
# uploaded = files.upload()
# D7data2.csv

import os
os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
os.getcwd()
```

```
'C:\\Users\\surya\\Downloads\\PG-DBDA-Mar23\\Datasets'
```

```
dataset = pd.read_csv('D7data2.csv')
dataset.head()
```

	AGE	WORKCLASS	FNLWGT	EDUCATION	EDUCATIONNUM	MARITALSTATUS	OCCUPATION	RELATIONSHI
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-famil
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husban
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-famil
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husban

```
dataset.shape
```

```
(32561, 15)
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   AGE            32561 non-null  int64 
 1   WORKCLASS      32561 non-null  object
 2   FNLWGT         32561 non-null  int64 
 3   EDUCATION      32561 non-null  object
 4   EDUCATIONNUM   32561 non-null  int64 
 5   MARITALSTATUS  32561 non-null  object
 6   OCCUPATION     32561 non-null  object
 7   RELATIONSHIP   32561 non-null  object
 8   RACE           32561 non-null  object
 9   SEX            32561 non-null  object
 10  CAPITALGAIN    32561 non-null  int64 
 11  CAPITALLOSS    32561 non-null  int64 
```

```
 12  HOURSPERWEEK   32561 non-null  int64
 13  NATIVECOUNTRY  32561 non-null  object
 14  ABOVE50K       32561 non-null  int64
dtypes: int64(7), object(8)
memory usage: 3.7+ MB
```

```
dataset.describe()
```

	AGE	FNLWGT	EDUCATIONNUM	CAPITALGAIN	CAPITALLOSS	HOURSPERWEEK
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

EDA

EDUCATION column

```
dataset['EDUCATION'].unique()
```

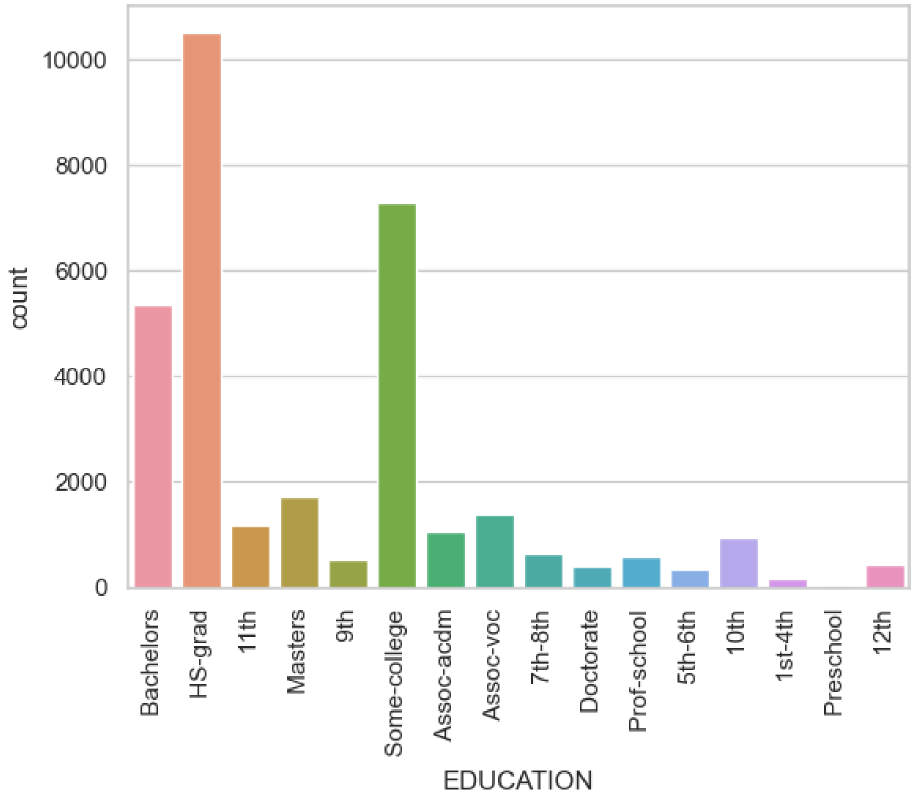
```
array([' Bachelors', ' HS-grad', ' 11th', ' Masters', ' 9th',
       ' Some-college', ' Assoc-acdm', ' Assoc-voc', ' 7th-8th',
       ' Doctorate', ' Prof-school', ' 5th-6th', ' 10th', ' 1st-4th',
       ' Preschool', ' 12th'], dtype=object)
```

```
dataset['EDUCATION'].nunique()
```

```
16
```

```
# plt.figure(figsize=(10, 10))
ax = sns.countplot(x=dataset['EDUCATION'], data=dataset, )
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
```

```
[Text(0, 0, ' Bachelors'),
 Text(1, 0, ' HS-grad'),
 Text(2, 0, ' 11th'),
 Text(3, 0, ' Masters'),
 Text(4, 0, ' 9th'),
 Text(5, 0, ' Some-college'),
 Text(6, 0, ' Assoc-acdm'),
 Text(7, 0, ' Assoc-voc'),
 Text(8, 0, ' 7th-8th'),
 Text(9, 0, ' Doctorate'),
 Text(10, 0, ' Prof-school'),
 Text(11, 0, ' 5th-6th'),
 Text(12, 0, ' 10th'),
 Text(13, 0, ' 1st-4th'),
 Text(14, 0, ' Preschool'),
 Text(15, 0, ' 12th')]
```



MARITALSTATUS column

```
dataset['MARITALSTATUS'].unique()
```

```
array([' Never-married', ' Married-civ-spouse', ' Divorced',
       ' Married-spouse-absent', ' Separated', ' Married-AF-spouse',
       ' Widowed'], dtype=object)
```

```
dataset['MARITALSTATUS'].nunique()
```

```
7
```

```
ax = sns.countplot(x=dataset['MARITALSTATUS'], data=dataset)
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
```

```
[Text(0, 0, ' Never-married'),
 Text(1, 0, ' Married-civ-spouse'),
 Text(2, 0, ' Divorced'),
 Text(3, 0, ' Married-spouse-absent'),
```

OCCUPATION column

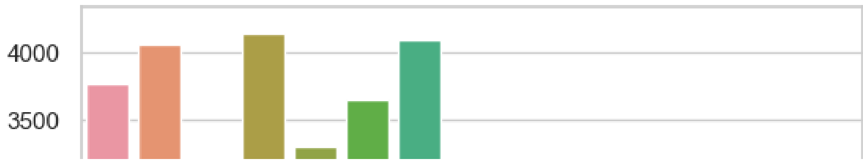
```
dataset['OCCUPATION'].unique()
```

```
array([' Adm-clerical', ' Exec-managerial', ' Handlers-cleaners',
       ' Prof-specialty', ' Other-service', ' Sales', ' Craft-repair',
       ' Transport-moving', ' Farming-fishing', ' Machine-op-inspct',
       ' Tech-support', ' ?', ' Protective-serv', ' Armed-Forces',
       ' Priv-house-serv'], dtype=object)
```

```
dataset['OCCUPATION'].nunique()
```

```
15
```

```
ax = sns.countplot(x=dataset['OCCUPATION'], data=dataset)
ax = ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
```



gender column


```
dataset['SEX'].unique()
```

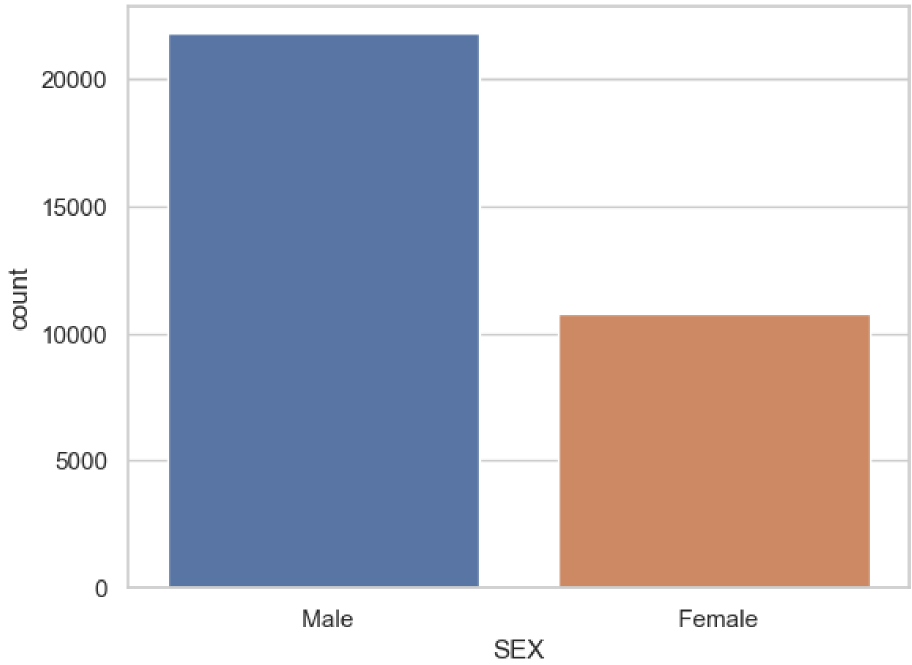
```
array([' Male', ' Female'], dtype=object)
```


```
dataset['SEX'].nunique()
```

```
2
```


```
sns.countplot(x = dataset['SEX'], data=dataset)
```

```
<Axes: xlabel='SEX', ylabel='count'>
```



RACE column

```
dataset['RACE'].unique()
```

```
array([' White', ' Black', ' Asian-Pac-Islander', ' Amer-Indian-Eskimo',
       ' Other'], dtype=object)
```

```
dataset['RACE'].nunique()
```

```
5
```

```
ax = sns.countplot(x = dataset['RACE'].unique(), data=dataset)
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
```

```
[Text(0, 0, ' White'),
```

AGE column

```
Text(4, 0, ' Other')]
```

```
dataset['AGE'].unique()
```

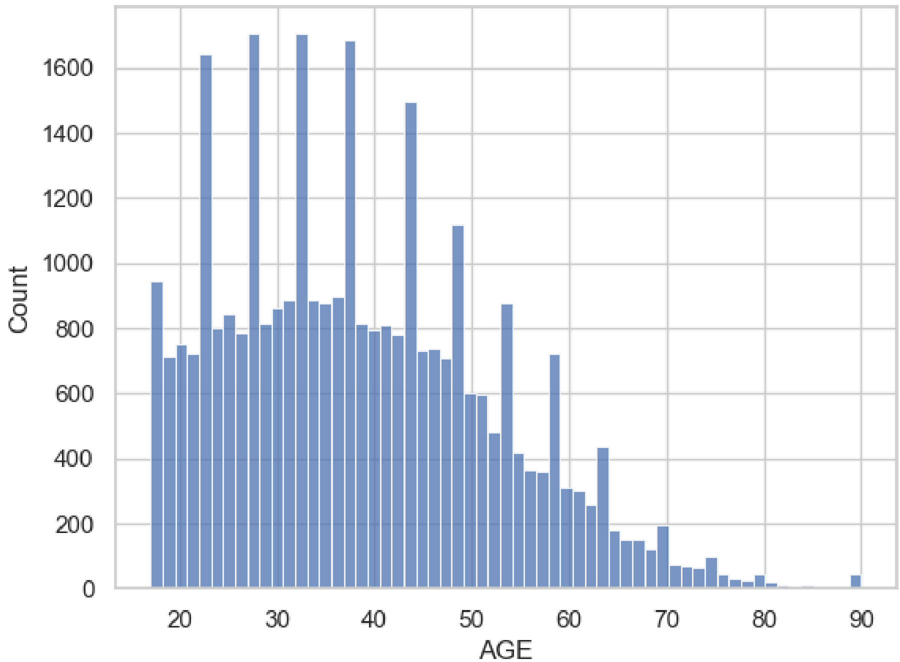
```
array([39, 50, 38, 53, 28, 37, 49, 52, 31, 42, 30, 23, 32, 40, 34, 25, 43,
       54, 35, 59, 56, 19, 20, 45, 22, 48, 21, 24, 57, 44, 41, 29, 18, 47,
       46, 36, 79, 27, 67, 33, 76, 17, 55, 61, 70, 64, 71, 68, 66, 51, 58,
       26, 60, 90, 75, 65, 77, 62, 63, 80, 72, 74, 69, 73, 81, 78, 88, 82,
       83, 84, 85, 86, 87], dtype=int64)
```

```
dataset['AGE'].nunique()
```

```
73
```

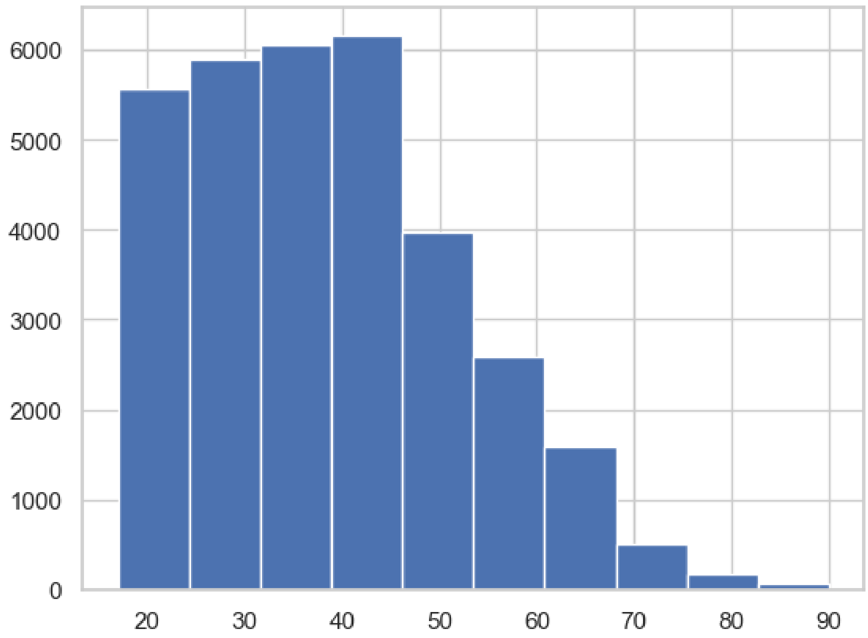
```
sns.histplot(x=dataset['AGE'], data=dataset)
```

```
<Axes: xlabel='AGE', ylabel='Count'>
```




```
# alt for histogram using pandas DataFrame
dataset.iloc[:, 0].hist()
```

```
<Axes: >
```



HOURSPERWEEK column

```
1 dataset['HOURSPERWEEK'].unique()
```

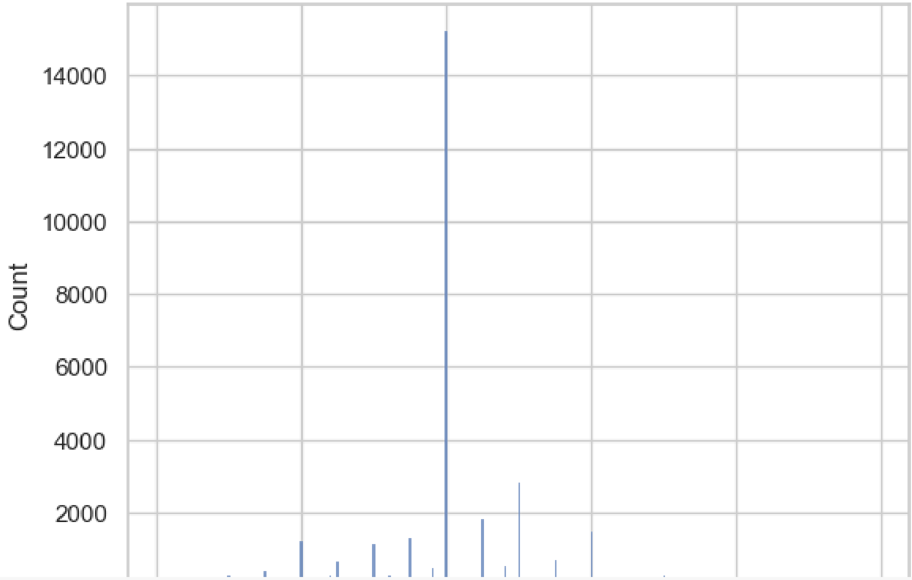
```
array([40, 13, 16, 45, 50, 80, 30, 35, 60, 20, 52, 44, 15, 25, 38, 43, 55,
       48, 58, 32, 70,  2, 22, 56, 41, 28, 36, 24, 46, 42, 12, 65,  1, 10,
       34, 75, 98, 33, 54,  8,  6, 64, 19, 18, 72,  5,  9, 47, 37, 21, 26,
       14,  4, 59,  7, 99, 53, 39, 62, 57, 78, 90, 66, 11, 49, 84,  3, 17,
       68, 27, 85, 31, 51, 77, 63, 23, 87, 88, 73, 89, 97, 94, 29, 96, 67,
       82, 86, 91, 81, 76, 92, 61, 74, 95], dtype=int64)
```

```
1 dataset['HOURSPERWEEK'].nunique()
```

```
94
```

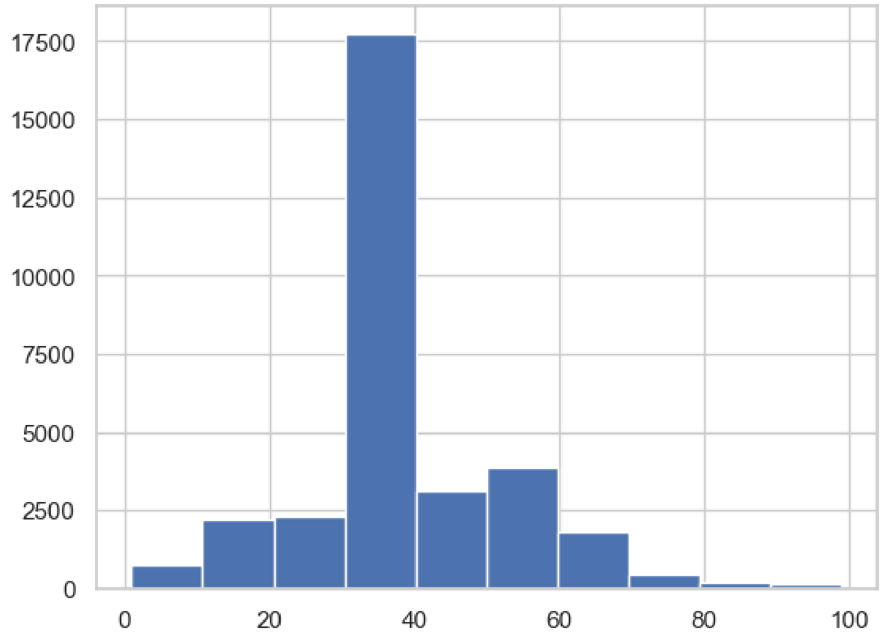
```
1 sns.histplot(x=dataset['HOURSPERWEEK'], data=dataset)
```

```
<Axes: xlabel='HOURSPERWEEK', ylabel='Count'>
```




```
# alt for histogram using pandas DataFrame
dataset.iloc[:, 12].hist()
```

```
<Axes: >
```



Null check

```
dataset.isnull().sum()
```

```
AGE              0
WORKCLASS        0
FNLWGT           0
EDUCATION        0
EDUCATIONNUM     0
MARITALSTATUS    0
OCCUPATION       0
RELATIONSHIP     0
RACE             0
SEX              0
CAPITALGAIN      0
CAPITALLOSS      0
HOURSPERWEEK     0
NATIVECOUNTRY    0
ABOVE50K         0
dtype: int64
```

dropping unwanted

```
dataset.drop(['FNLWGT', 'EDUCATION', 'MARITALSTATUS', 'RELATIONSHIP', 'CAPITALGAIN', 'CAPITALLOSS', 'NATIVECOUNTRY'], axis=1, inplace=True)
```

```
dataset.head()
```

	AGE	WORKCLASS	EDUCATIONNUM	OCCUPATION	RACE	SEX	HOURSPERWEEK	ABOVE50K
0	39	State-gov	13	Adm-clerical	White	Male	40	0
1	50	Self-emp-not-inc	13	Exec-managerial	White	Male	13	0
2	38	Private	9	Handlers-cleaners	White	Male	40	0
3	53	Private	7	Handlers-cleaners	Black	Male	40	0

```
dataset.shape
```

```
(32561, 8)
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   AGE           32561 non-null  int64 
 1   WORKCLASS     32561 non-null  object
 2   EDUCATIONNUM  32561 non-null  int64 
 3   OCCUPATION    32561 non-null  object
 4   RACE          32561 non-null  object
 5   SEX           32561 non-null  object
 6   HOURSPERWEEK  32561 non-null  int64 
 7   ABOVE50K      32561 non-null  int64 
dtypes: int64(4), object(4)
memory usage: 2.0+ MB
```

identify X & Y

```
# independent variables
x = dataset.iloc[ : , 6].values
x[:5]
```

```
array([40, 13, 40, 40, 40], dtype=int64)
```

```
# dependent variables
y = dataset.iloc[ : , 7].values
y[:5]
```

```
array([0, 0, 0, 0, 0], dtype=int64)
```

Splitting

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

```
x_train[:5]
```

```
array([40, 54, 50, 40, 40], dtype=int64)
```

```
x_test[:5]
```

```
array([44, 40, 40, 40, 76], dtype=int64)
```

```
y_train[:5]
```

```
array([0, 0, 0, 0, 0], dtype=int64)
```

```
y_test[:5]
```

```
array([0, 0, 0, 0, 1], dtype=int64)
```

Preprocessing

reshaping

- need to reshape X as it is just single column, so need to reshape from horizontal shape to vertical shape

```
x_train = x_train.reshape(-1, 1)
x_train[:5]
```

```
array([[40],
       [54],
       [50],
       [40],
       [40]], dtype=int64)
```

```
x_test = x_test.reshape(-1, 1)
x_test[:5]
```

```
array([[44],
       [40],
       [40],
       [40],
       [76]], dtype=int64)
```

Modeling

```
from sklearn.linear_model import LogisticRegression
```

```
model = LogisticRegression()
```

Training

```
model.fit(x_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

Predict

```
lr_model_pred = model.predict(x_test)
lr_model_pred[:5]
```

```
array([0, 0, 0, 0, 1], dtype=int64)
```

Evaluation

accuracy_score

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(y_test, lr_model_pred)
```

```
0.7466605251036389
```

precision_score

```
from sklearn.metrics import precision_score
```

```
precision_score(y_test, lr_model_pred)
```

```
0.35135135135135137
```

recall_score

```
from sklearn.metrics import recall_score
```

```
recall_score(y_test, lr_model_pred)
```

```
0.04075235109717868
```

f1_score

```
from sklearn.metrics import f1_score
```

```
f1_score(y_test, lr_model_pred)
```

```
0.07303370786516854
```

classification_report

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, lr_model_pred))
```

```
              precision    recall  f1-score   support

           0       0.76      0.98      0.85      4918
           1       0.35      0.04      0.07      1595

    accuracy                           0.75      6513
   macro avg       0.55      0.51      0.46      6513
weighted avg       0.66      0.75      0.66      6513
```

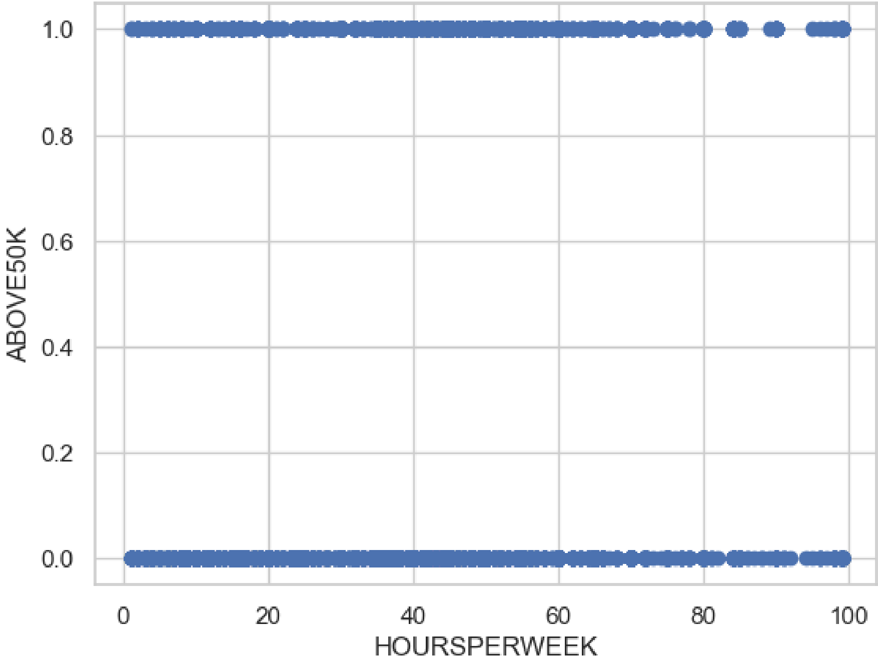
Visualize classification

scatter()

- for classification

```
plt.scatter(dataset['HOURSPERWEEK'], dataset['ABOVE50K'])
plt.xlabel('HOURSPERWEEK')
plt.ylabel('ABOVE50K')
# shows data point classified as 0 or 1
```

```
Text(0, 0.5, 'ABOVE50K')
```

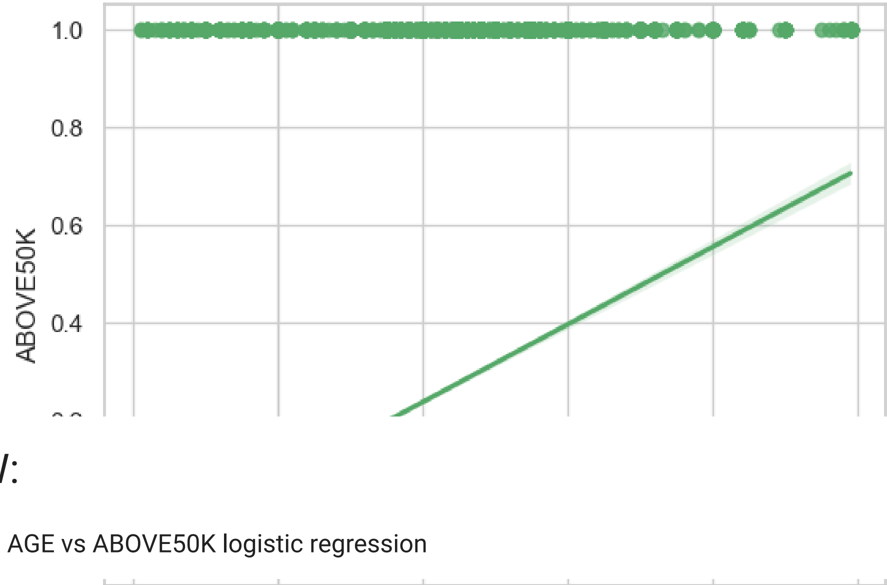


regplot()

- Regression Line Plot

```
sns.regplot(x='HOURSPERWEEK', y='ABOVE50K', color='g', data=dataset)
# shows regression is converte into classification
```

```
<Axes: xlabel='HOURSPERWEEK', ylabel='ABOVE50K'>
```



HW:

- AGE vs ABOVE50K logistic regression

```
1
```