ML - Lab - Assignment 2
Data Preprocessing

[204]:
```python
import pandas as pd
import numpy as np
```

[205]:
```python
df = pd.read_csv('D:\MIT ADT\Third Year - Sem 2\ML LAB\Assign 2\\adult.csv')
```

[206]:
```python
df.shape
```

[206]:
```
(48842, 15)
```

[207]:
```python
df.head()
```

[207]:
```
   age workclass  fnlwgt     education  educational-num      marital-status  \
0   25   Private  226802          11th                7       Never-married
1   38   Private   89814       HS-grad                9  Married-civ-spouse
2   28 Local-gov  336951    Assoc-acdm               12  Married-civ-spouse
3   44   Private  160323  Some-college               10  Married-civ-spouse
4   18         ?  103497  Some-college               10       Never-married

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

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

[208]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
```

```
---  ------          --------------  -----
 0   age             48842 non-null  int64
 1   workclass       48842 non-null  object
 2   fnlwgt          48842 non-null  int64
 3   education       48842 non-null  object
 4   educational-num 48842 non-null  int64
 5   marital-status  48842 non-null  object
 6   occupation      48842 non-null  object
 7   relationship    48842 non-null  object
 8   race            48842 non-null  object
 9   gender          48842 non-null  object
 10  capital-gain    48842 non-null  int64
 11  capital-loss    48842 non-null  int64
 12  hours-per-week  48842 non-null  int64
 13  native-country  48842 non-null  object
 14  income          48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

[209]:
```
df.describe()
```

[209]:
```
                age        fnlwgt  educational-num  capital-gain  \
count  48842.000000  4.884200e+04     48842.000000  48842.000000
mean      38.643585  1.896641e+05        10.078089   1079.067626
std       13.710510  1.056040e+05         2.570973   7452.019058
min       17.000000  1.228500e+04         1.000000      0.000000
25%       28.000000  1.175505e+05         9.000000      0.000000
50%       37.000000  1.781445e+05        10.000000      0.000000
75%       48.000000  2.376420e+05        12.000000      0.000000
max       90.000000  1.490400e+06        16.000000  99999.000000

       capital-loss  hours-per-week
count  48842.000000    48842.000000
mean      87.502314       40.422382
std      403.004552       12.391444
min        0.000000        1.000000
25%        0.000000       40.000000
50%        0.000000       40.000000
75%        0.000000       45.000000
max     4356.000000       99.000000
```

[210]:
```
df.isna().sum()
```

[210]:
```
age                0
workclass          0
fnlwgt             0
education          0
```

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

[211]:
```python
df.duplicated().sum()
```

[211]:
```
52
```

[212]:
```python
df=df.drop_duplicates()
```

[213]:
```python
df.isin(['?']).sum()
```

[213]:
```
age                   0
workclass          2795
fnlwgt                0
education             0
educational-num       0
marital-status        0
occupation         2805
relationship          0
race                  0
gender                0
capital-gain          0
capital-loss          0
hours-per-week        0
native-country      856
income                0
dtype: int64
```

1 Handling Missing Values

1. Leave as it is
2. Fill the missing values
3. Drop missing values

[214]:
```python
df = df.replace('?',np.nan)
```

[215]:
```
df.isna().sum()
```

[215]:
```
age                  0
workclass         2795
fnlwgt               0
education            0
educational-num      0
marital-status       0
occupation        2805
relationship         0
race                 0
gender               0
capital-gain         0
capital-loss         0
hours-per-week       0
native-country     856
income               0
dtype: int64
```

2 Drop missing values

3 DataFrameName.dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)

axis: axis takes int or string value for rows/columns. Input can be 0 or 1 for Integer and 'index' or 'columns' for String. how: how takes string value of two kinds only ('any' or 'all'). 'any' drops the row/column if ANY value is Null and 'all' drops only if ALL values are null. thresh: thresh takes integer value which tells minimum amount of na values to drop. subset: It's an array which limits the dropping process to passed rows/columns through list. inplace: It is a boolean which makes the changes in data frame itself if True.

[216]:
```
temp = pd.DataFrame({
    "Name":['Abc',"PQR", np.nan,"XYZ"],
    "Roll No":[1,np.nan,3,4]
})
```

[217]:
```
temp
```

[217]:

	Name	Roll No
0	Abc	1.0
1	PQR	NaN
2	NaN	3.0
3	XYZ	4.0

[218]:
```
temp=temp.dropna(inplace=False)
```

[219]:
```
temp
```

[219]:

	Name	Roll No
0	Abc	1.0
3	XYZ	4.0

4 Fill the rows with missing values

5 DataFrame.fillna(value=None, method=None, axis=None, inplace=False, limit=None, downcast=None, **kwargs)

value : Static, dictionary, array, series or dataframe to fill instead of NaN. method : Method is used if user doesn't pass any value. Pandas has different methods like bfill, backfill or ffill which fills the place with value in the Forward index or Previous/Back respectively. axis: axis takes int or string value for rows/columns. Input can be 0 or 1 for Integer and 'index' or 'columns' for String inplace: It is a boolean which makes the changes in data frame itself if True. limit : This is an integer value which specifies maximum number of consecutive forward/backward NaN value fills. downcast : It takes a dict which specifies what dtype to downcast to which one. Like Float64 to int64. **kwargs : Any other Keyword arguments

[220]:
```
temp
```

[220]:

	Name	Roll No
0	Abc	1.0
3	XYZ	4.0

[221]:
```
temp.fillna(method='ffill',inplace=True)
```

```
/var/folders/yh/sv7lkgq112103y8hwl95rv1r0000gn/T/ipykernel_2450/2967702086.py:1:
FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a
future version. Use obj.ffill() or obj.bfill() instead.
  temp.fillna(method='ffill',inplace=True)
/var/folders/yh/sv7lkgq112103y8hwl95rv1r0000gn/T/ipykernel_2450/2967702086.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  temp.fillna(method='ffill',inplace=True)
```

[222]:
```
temp
```

[222]:

	Name	Roll No
0	Abc	1.0
3	XYZ	4.0

6 Simple Imputer

It replaces the NaN values with a specified placeholder.

missing_values : The missing_values placeholder which has to be imputed. By default is NaN strategy : The data which will replace the NaN values from the dataset. The strategy argument can take the values – 'mean'(default), 'median', 'most_frequent' and 'constant'. fill_value : The constant value to be given to the NaN data using the constant strategy.

[223]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='most_frequent', missing_values=np.nan)

df['workclass'] = imputer.fit_transform(df[['workclass']]).ravel()
df['occupation'] = imputer.fit_transform(df[['occupation']]).ravel()
df['native-country'] = imputer.fit_transform(df[['native-country']]).ravel()
```

[224]:
```
df.isna().sum()
```

[224]:
```
age                0
workclass          0
fnlwgt             0
education          0
educational-num    0
marital-status     0
occupation         0
relationship       0
race               0
gender             0
capital-gain       0
capital-loss       0
hours-per-week     0
native-country     0
income             0
dtype: int64
```

[225]:
```
df['gender'].unique()
```

[225]:
```
array(['Male', 'Female'], dtype=object)
```

[226]:
```
df
```

[226]:
```
       age   workclass  fnlwgt     education  educational-num  \
0       25     Private  226802          11th                7
1       38     Private   89814       HS-grad                9
2       28   Local-gov  336951    Assoc-acdm               12
3       44     Private  160323  Some-college               10
4       18     Private  103497  Some-college               10
…     …            …       …             …              …
```

```
48837   27        Private  257302     Assoc-acdm              12
48838   40        Private  154374        HS-grad               9
48839   58        Private  151910        HS-grad               9
48840   22        Private  201490        HS-grad               9
48841   52   Self-emp-inc  287927        HS-grad               9

           marital-status         occupation relationship   race  gender  \
0           Never-married  Machine-op-inspct    Own-child  Black    Male
1      Married-civ-spouse    Farming-fishing      Husband  White    Male
2      Married-civ-spouse    Protective-serv      Husband  White    Male
3      Married-civ-spouse  Machine-op-inspct      Husband  Black    Male
4           Never-married     Prof-specialty    Own-child  White  Female
…                     …                  …          …      …       …
48837  Married-civ-spouse       Tech-support         Wife  White  Female
48838  Married-civ-spouse  Machine-op-inspct      Husband  White    Male
48839             Widowed       Adm-clerical    Unmarried  White  Female
48840       Never-married       Adm-clerical    Own-child  White    Male
48841  Married-civ-spouse    Exec-managerial         Wife  White  Female

       capital-gain  capital-loss  hours-per-week native-country income
0                 0             0              40  United-States  <=50K
1                 0             0              50  United-States  <=50K
2                 0             0              40  United-States   >50K
3              7688             0              40  United-States   >50K
4                 0             0              30  United-States  <=50K
…               …             …              …            …     …
48837             0             0              38  United-States  <=50K
48838             0             0              40  United-States   >50K
48839             0             0              40  United-States  <=50K
48840             0             0              20  United-States  <=50K
48841         15024             0              40  United-States   >50K

[48790 rows x 15 columns]
```

7 Handling Categorical Data

1. Replace
2. Label Encoding
3. One hot Encoding

[227]:
```
print(df['gender'].unique())
```

```
['Male' 'Female']
```

[228]:
```
df['gender']=df['gender'].replace('Male',1)
df['gender']=df['gender'].replace('Female',0)
```

```
/var/folders/yh/sv7lkgq112103y8hwl95rv1r0000gn/T/ipykernel_2450/752269921.py:2:
FutureWarning: Downcasting behavior in `replace` is deprecated and will be
removed in a future version. To retain the old behavior, explicitly call
`result.infer_objects(copy=False)`. To opt-in to the future behavior, set
`pd.set_option('future.no_silent_downcasting', True)`
  df['gender']=df['gender'].replace('Female',0)
```

[229]:
```python
temp_df = pd.DataFrame({
    'Fruit_Name':['Mango','Apple','Banana','Grapes'],
    'Fruit_Color':['Yellow','Red','Yellow','Green'],
    'Fruit_Price':[1000,300,50,100]
})
```

[230]:
```python
temp_df
```

[230]:
```
  Fruit_Name Fruit_Color  Fruit_Price
0      Mango      Yellow         1000
1      Apple         Red          300
2     Banana      Yellow           50
3     Grapes       Green          100
```

[231]:
```python
from sklearn.preprocessing import LabelEncoder
lbl_encoder=LabelEncoder()
temp_df['Fruit_Name']=lbl_encoder.fit_transform(temp_df['Fruit_Name'])
```

[232]:
```python
temp_df
```

[232]:
```
   Fruit_Name Fruit_Color  Fruit_Price
0           3      Yellow         1000
1           0         Red          300
2           1      Yellow           50
3           2       Green          100
```

[233]:
```python
temp_df=pd.get_dummies(temp_df,columns=['Fruit_Color'])
```

[234]:
```python
temp_df
```

[234]:
```
   Fruit_Name  Fruit_Price  Fruit_Color_Green  Fruit_Color_Red  \
0           3         1000              False            False
1           0          300              False             True
2           1           50              False            False
3           2          100               True            False

   Fruit_Color_Yellow
0                True
1               False
2                True
```

```
3                False
```

[235]:
```python
print(df['income'].unique())
```

```
['<=50K' '>50K']
```

[236]:
```python
df['income']=df['income'].replace('<=50K',0)
df['income']=df['income'].replace('>50K',1)
```

```
/var/folders/yh/sv7lkgq112103y8hwl95rv1r0000gn/T/ipykernel_2450/3062699630.py:2:
FutureWarning: Downcasting behavior in `replace` is deprecated and will be
removed in a future version. To retain the old behavior, explicitly call
`result.infer_objects(copy=False)`. To opt-in to the future behavior, set
`pd.set_option('future.no_silent_downcasting', True)`
  df['income']=df['income'].replace('>50K',1)
```

[237]:
```python
print(df['income'].unique())
```

```
[0 1]
```

[238]:
```python
print(df['marital-status'].unique())
```

```
['Never-married' 'Married-civ-spouse' 'Widowed' 'Divorced' 'Separated'
 'Married-spouse-absent' 'Married-AF-spouse']
```

[239]:
```python
df['marital-status'] = df['marital-status'].replace('Never-married',␣
  ↪'Unmarried')
df['marital-status'] = df['marital-status'].replace('Married-AF-spouse',␣
  ↪'Married')
df['marital-status'] = df['marital-status'].replace('Married-civ-spouse',␣
  ↪'Married')
df['marital-status'] = df['marital-status'].replace('Married-spouse-absent',␣
  ↪'Unmarried')
df['marital-status'] = df['marital-status'].replace('Separated', 'Separated')
df['marital-status'] = df['marital-status'].replace('Divorced', 'Separated')
df['marital-status'] = df['marital-status'].replace('Widowed', 'Widowed')
```

[240]:
```python
df['marital-status']=lbl_encoder.fit_transform(df['marital-status'])
```

[241]:
```python
print(df['marital-status'].unique())
```

```
[2 0 3 1]
```

[242]:
```python
df['education'].unique()
```

[242]:
```
array(['11th', 'HS-grad', 'Assoc-acdm', 'Some-college', '10th',
       'Prof-school', '7th-8th', 'Bachelors', 'Masters', 'Doctorate',
       '5th-6th', 'Assoc-voc', '9th', '12th', '1st-4th', 'Preschool'],
```

```
      dtype=object)
```

[243]:
```
df['education'] = df['education'].replace('Preschool', 'dropout')
df['education'] = df['education'].replace('10th', 'dropout')
df['education'] = df['education'].replace('11th', 'dropout')
df['education'] = df['education'].replace('12th', 'dropout')
df['education'] = df['education'].replace('1st-4th', 'dropout')
df['education'] = df['education'].replace('5th-6th', 'dropout')
df['education'] = df['education'].replace('7th-8th', 'dropout')
df['education'] = df['education'].replace('9th', 'dropout')
df['education'] = df['education'].replace('HS-Grad', 'HighGrad')
df['education'] = df['education'].replace('HS-grad', 'HighGrad')
df['education'] = df['education'].replace('Some-college', 'CommunityCollege')
df['education'] = df['education'].replace('Assoc-acdm', 'CommunityCollege')
df['education'] = df['education'].replace('Assoc-voc', 'CommunityCollege')
df['education'] = df['education'].replace('Bachelors', 'Bachelors')
df['education'] = df['education'].replace('Masters', 'Masters')
df['education'] = df['education'].replace('Prof-school', 'Masters')
df['education'] = df['education'].replace('Doctorate', 'Doctorate')
```

[244]:
```
df['education'].unique()
```

[244]:
```
array(['dropout', 'HighGrad', 'CommunityCollege', 'Masters', 'Bachelors',
       'Doctorate'], dtype=object)
```

[245]:
```
df['education']=lbl_encoder.fit_transform(df['education'])
```

[246]:
```
df['workclass'] = lbl_encoder.fit_transform(df['workclass'])
df['occupation'] = lbl_encoder.fit_transform(df['occupation'])
df['relationship'] = lbl_encoder.fit_transform(df['relationship'])
df['race'] = lbl_encoder.fit_transform(df['race'])
df['native-country'] = lbl_encoder.fit_transform(df['native-country'])
```

[247]:
```
df
```

[247]:

	age	workclass	fnlwgt	education	educational-num	marital-status	\
0	25	3	226802	5	7	2	
1	38	3	89814	3	9	0	
2	28	1	336951	1	12	0	
3	44	3	160323	1	10	0	
4	18	3	103497	1	10	2	
…	…	…	…	…	…	…	
48837	27	3	257302	1	12	0	
48838	40	3	154374	3	9	0	
48839	58	3	151910	3	9	3	
48840	22	3	201490	3	9	2	
48841	52	4	287927	3	9	0	

```
       occupation  relationship  race  gender  capital-gain  capital-loss  \
0               6             3     2       1             0             0
1               4             0     4       1             0             0
2              10             0     4       1             0             0
3               6             0     2       1          7688             0
4               9             3     4       0             0             0
…             …             …   …     …           …           …
48837          12             5     4       0             0             0
48838           6             0     4       1             0             0
48839           0             4     4       0             0             0
48840           0             3     4       1             0             0
48841           3             5     4       0         15024             0

       hours-per-week  native-country  income
0                  40              38       0
1                  50              38       0
2                  40              38       1
3                  40              38       1
4                  30              38       0
…               …               …     …
48837              38              38       0
48838              40              38       1
48839              40              38       0
48840              20              38       0
48841              40              38       1

[48790 rows x 15 columns]
```

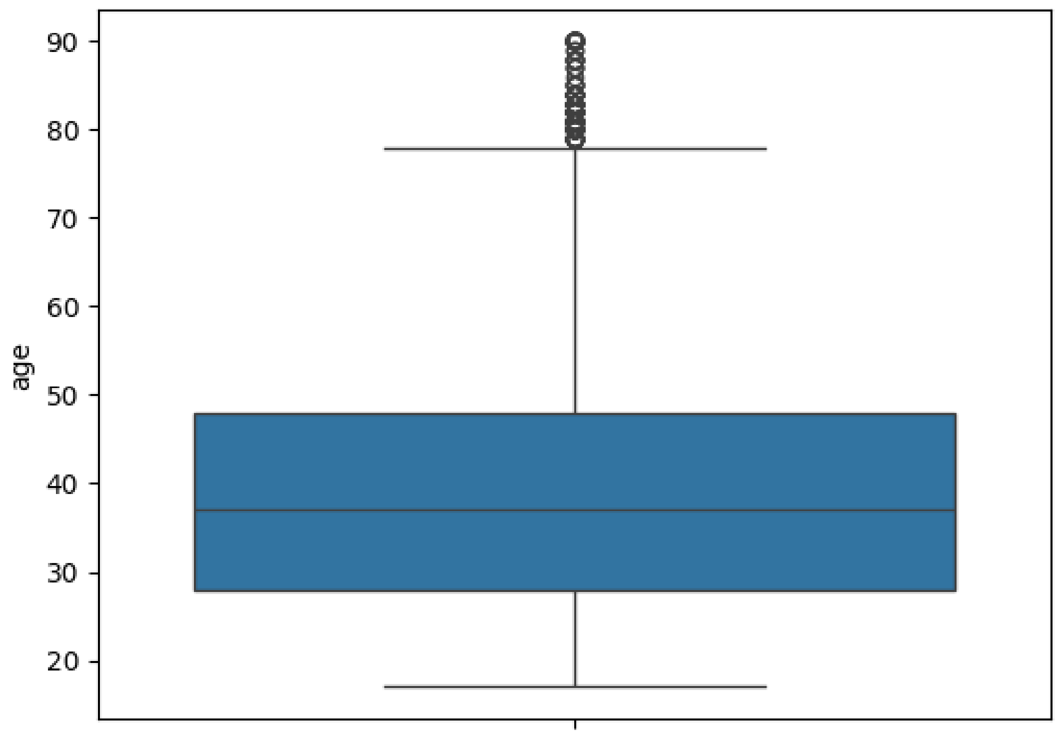
8 Outlier Detection

1. Use BoxPlot
2. Use Scatter Plot
3. Use Z score
4. Inter Quartile Range

[248]:
```
import seaborn as sns
sns.boxplot(df['age'])
```

```
/Users/nageshjadhav/miniforge3/lib/python3.10/site-
packages/seaborn/categorical.py:640: FutureWarning: SeriesGroupBy.grouper is
deprecated and will be removed in a future version of pandas.
  positions = grouped.grouper.result_index.to_numpy(dtype=float)
```

[248]:
```
<Axes: ylabel='age'>
```



[249]:
```
print(df['age'].sort_values().unique())
```

```
[17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88
 89 90]
```

[250]:
```
test= np.where(df['age']>78)
print(test)
```

```
(array([  193,   234,   898,   925,   950,  1078,  1397,  1833,  2084,
         2289,  2981,  3495,  3667,  4454,  4645,  4657,  6401,  6576,
         6756,  6914,  6958,  6975,  6978,  7159,  7169,  7413,  7418,
         7538,  7546,  7936,  8205,  8312,  8426,  8954,  8981,  9017,
         9037,  9080,  9278,  9768,  9887, 10038, 10198, 10222, 10734,
        11286, 11325, 11407, 11834, 11868, 11878, 11937, 12057, 12226,
        12443, 13022, 13954, 14029, 14259, 14295, 14427, 14564, 14587,
        14736, 15084, 15094, 15404, 15930, 15958, 15998, 16101, 16143,
        16246, 16350, 16498, 16706, 17194, 17316, 17444, 18211, 18578,
        19029, 19166, 19181, 19485, 19612, 19810, 20050, 20236, 20343,
        20382, 20992, 21106, 21542, 21561, 21640, 21676, 22270, 22443,
        22484, 22502, 22709, 22894, 23018, 23751, 23990, 24142, 24445,
        24650, 24700, 24791, 24963, 25075, 25231, 25241, 25737, 26388,
        26474, 26809, 27363, 27502, 27776, 27796, 27994, 28259, 28714,
```

```
28755, 29093, 29238, 29288, 29289, 29557, 29958, 30190, 30366,
30421, 30865, 30971, 31016, 31163, 31615, 31921, 32151, 32561,
32782, 33021, 33160, 33867, 34294, 34398, 34529, 34534, 34670,
34816, 34980, 35087, 35300, 35427, 35435, 35467, 35744, 35750,
35770, 35943, 36001, 36082, 36503, 36675, 36717, 36736, 36737,
36862, 37078, 37132, 37205, 37594, 37751, 38062, 38085, 38468,
38726, 39139, 39142, 39703, 40144, 40271, 40287, 40482, 40524,
40639, 40804, 41406, 41546, 41640, 42253, 42483, 42971, 44035,
44415, 44701, 44958, 45184, 45829, 45959, 47261, 47663, 47927,
48045, 48067, 48086, 48507, 48597, 48688, 48723, 48754]),)
```

[251]:
```
sorted_df = df.sort_values(by=['age'],ascending=True)

Q1=np.percentile(sorted_df['age'],25)

Q3=np.percentile(sorted_df['age'],75)

IQR = Q3-Q1
print(IQR)
```

```
20.0
```

[252]:
```
lwr_bound = Q1-(1.5*IQR)
upr_bound = Q3+(1.5*IQR)
print("min: ", lwr_bound, " Max: ", upr_bound)
```

```
min:  -2.0  Max:  78.0
```

[253]:
```
outliers=[]
for i in df['age']:
    if(i<lwr_bound or i>upr_bound):
        outliers.append(i)
print("No. of outliers: ",len(outliers))
print(outliers)
```

```
No. of outliers:  215
[79, 80, 90, 79, 80, 81, 82, 83, 81, 85, 80, 90, 81, 84, 81, 89, 81, 83, 81, 82,
80, 90, 81, 83, 80, 90, 90, 84, 80, 80, 80, 81, 90, 85, 90, 81, 81, 80, 80, 79,
81, 80, 88, 87, 90, 79, 83, 79, 80, 90, 79, 79, 81, 81, 90, 82, 90, 87, 81, 88,
80, 81, 80, 81, 90, 88, 89, 84, 80, 80, 83, 79, 81, 79, 90, 80, 81, 90, 88, 90,
90, 80, 90, 81, 82, 79, 81, 80, 83, 90, 90, 79, 81, 90, 80, 90, 90, 79, 79, 84,
90, 80, 90, 81, 83, 84, 81, 79, 85, 82, 79, 80, 90, 90, 90, 84, 80, 90, 90, 79,
84, 90, 79, 90, 90, 90, 82, 81, 90, 84, 79, 81, 82, 81, 80, 90, 80, 84, 82, 79,
90, 84, 90, 83, 79, 81, 80, 79, 80, 79, 80, 90, 90, 80, 90, 90, 81, 83, 82, 90,
90, 81, 80, 80, 90, 79, 80, 82, 85, 80, 79, 90, 81, 79, 80, 79, 81, 82, 88, 90,
82, 88, 84, 83, 79, 86, 90, 90, 82, 83, 81, 79, 90, 80, 81, 79, 84, 84, 79, 90,
80, 81, 81, 81, 90, 87, 90, 80, 80, 82, 90, 90, 85, 82, 81]
```

9 Handling Outliers

1. Removing outliers
2. Quartile based flooring and capping
3. Mean/Meadian Imputation

[254]:
```
median=np.median(df['age'])
print(median)
for i in outliers:
    df['age'] = np.where(df['age']==i,37,df['age'])
```

```
37.0
```

[255]:
```
df['age'].sort_values().unique()
```

[255]:
```
array([17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78])
```

10 Data Sampling

[256]:
```
df['income'].value_counts()
```

[256]:
```
income
0    37109
1    11681
Name: count, dtype: int64
```

[257]:
```
# Random Sampling
lt_fifty_k=df[df['income']==0]
gt_fifty_k=df[df['income']==1]

print("<=50k: ", lt_fifty_k.shape)
print(">50k: ", gt_fifty_k.shape)
```

```
<=50k:  (37109, 15)
>50k:  (11681, 15)
```

[258]:
```
no_sample=lt_fifty_k.sample(n=11681)
```

[259]:
```
no_sample.shape
```

[259]:
```
(11681, 15)
```

[260]:
```
sampled_df=pd.concat([no_sample,gt_fifty_k],axis=0)
```

[261]:
```
sampled_df.shape
```

[261]:
```
(23362, 15)
```

[262]:
```
df
```

[262]:
```
       age  workclass  fnlwgt  education  educational-num  marital-status  \
0       25          3  226802          5                7               2
1       38          3   89814          3                9               0
2       28          1  336951          1               12               0
3       44          3  160323          1               10               0
4       18          3  103497          1               10               2
…      …          …       …          …                …               …
48837   27          3  257302          1               12               0
48838   40          3  154374          3                9               0
48839   58          3  151910          3                9               3
48840   22          3  201490          3                9               2
48841   52          4  287927          3                9               0

       occupation  relationship  race  gender  capital-gain  capital-loss  \
0               6             3     2       1             0             0
1               4             0     4       1             0             0
2              10             0     4       1             0             0
3               6             0     2       1          7688             0
4               9             3     4       0             0             0
…             …             …    …      …           …           …
48837          12             5     4       0             0             0
48838           6             0     4       1             0             0
48839           0             4     4       0             0             0
48840           0             3     4       1             0             0
48841           3             5     4       0         15024             0

       hours-per-week  native-country  income
0                  40              38       0
1                  50              38       0
2                  40              38       1
3                  40              38       1
4                  30              38       0
…                 …             …      …
48837              38              38       0
48838              40              38       1
48839              40              38       0
48840              20              38       0
48841              40              38       1

[48790 rows x 15 columns]
```

[263]:
```python
X = df.drop('income',axis=1)
y = df['income']
```

[264]:
```python
print("Shape of X: ", X.shape)
print("Shape of y: ", y.shape)
```

```
Shape of X:  (48790, 14)
Shape of y:  (48790,)
```

[265]:
```python
df.corr()
```

[265]:
```
                     age  workclass    fnlwgt  education  educational-num  \
age             1.000000   0.044471 -0.073595   0.063870         0.036319
workclass       0.044471   1.000000 -0.026516   0.011353         0.007331
fnlwgt         -0.073595  -0.026516  1.000000   0.019093        -0.038727
education       0.063870   0.011353  0.019093   1.000000        -0.605655
educational-num 0.036319   0.007331 -0.038727  -0.605655         1.000000
marital-status -0.341045  -0.054321  0.023984   0.008609        -0.084470
occupation     -0.002090   0.009878 -0.002391   0.006641         0.072688
relationship   -0.265529  -0.056085  0.009017   0.021186        -0.090697
race            0.027914   0.053939 -0.027165  -0.020251         0.029331
gender          0.089179   0.066675  0.027879   0.033225         0.009364
capital-gain    0.077915   0.031554 -0.003715  -0.006344         0.125219
capital-loss    0.056658   0.004160 -0.004378  -0.024376         0.080986
hours-per-week  0.087974   0.042887 -0.013521  -0.060474         0.143915
native-country -0.002915  -0.004872 -0.058299  -0.081753         0.089359
income          0.238085  -0.000508 -0.006309  -0.134587         0.332802

                marital-status  occupation  relationship      race    gender  \
age                  -0.341045   -0.002090     -0.265529  0.027914  0.089179
workclass            -0.054321    0.009878     -0.056085  0.053939  0.066675
fnlwgt                0.023984   -0.002391      0.009017 -0.027165  0.027879
education             0.008609    0.006641      0.021186 -0.020251  0.033225
educational-num      -0.084470    0.072688     -0.090697  0.029331  0.009364
marital-status        1.000000    0.001833      0.450723 -0.086343 -0.378383
occupation            0.001833    1.000000     -0.035054 -0.005158  0.042773
relationship          0.450723   -0.035054      1.000000 -0.116985 -0.579955
race                 -0.086343   -0.005158     -0.116985  1.000000  0.086959
gender               -0.378383    0.042773     -0.579955  0.086959  1.000000
capital-gain         -0.079394    0.014498     -0.056543  0.011610  0.047127
capital-loss         -0.069299    0.011048     -0.057243  0.018640  0.045517
hours-per-week       -0.246011   -0.015454     -0.250319  0.039759  0.228529
native-country        0.001027   -0.001643     -0.007092  0.117740 -0.002544
income               -0.415356    0.032533     -0.253175  0.070970  0.214639

                capital-gain  capital-loss  hours-per-week  native-country  \
age                 0.077915      0.056658        0.087974       -0.002915
```

```
workclass             0.031554      0.004160       0.042887       -0.004872
fnlwgt               -0.003715     -0.004378      -0.013521       -0.058299
education            -0.006344     -0.024376      -0.060474       -0.081753
educational-num       0.125219      0.080986       0.143915        0.089359
marital-status       -0.079394     -0.069299      -0.246011        0.001027
occupation            0.014498      0.011048      -0.015454       -0.001643
relationship         -0.056543     -0.057243      -0.250319       -0.007092
race                  0.011610      0.018640       0.039759        0.117740
gender                0.047127      0.045517       0.228529       -0.002544
capital-gain          1.000000     -0.031475       0.082152        0.007884
capital-loss         -0.031475      1.000000       0.054431        0.006466
hours-per-week        0.082152      0.054431       1.000000        0.006668
native-country        0.007884      0.006466       0.006668        1.000000
income                0.223047      0.147542       0.227664        0.020161

                   income
age              0.238085
workclass       -0.000508
fnlwgt          -0.006309
education       -0.134587
educational-num  0.332802
marital-status  -0.415356
occupation       0.032533
relationship    -0.253175
race             0.070970
gender           0.214639
capital-gain     0.223047
capital-loss     0.147542
hours-per-week   0.227664
native-country   0.020161
income           1.000000
```

[266]:
```python
from sklearn.feature_selection import mutual_info_classif
# determine the mutual information
mutual_info = mutual_info_classif(X, y)
mutual_info
```

[266]:
```
array([0.06571307, 0.01826695, 0.03867816, 0.06479106, 0.06558839,
       0.10961218, 0.0538709 , 0.11534223, 0.01391617, 0.03003066,
       0.08205115, 0.03504005, 0.0425127 , 0.00843042])
```

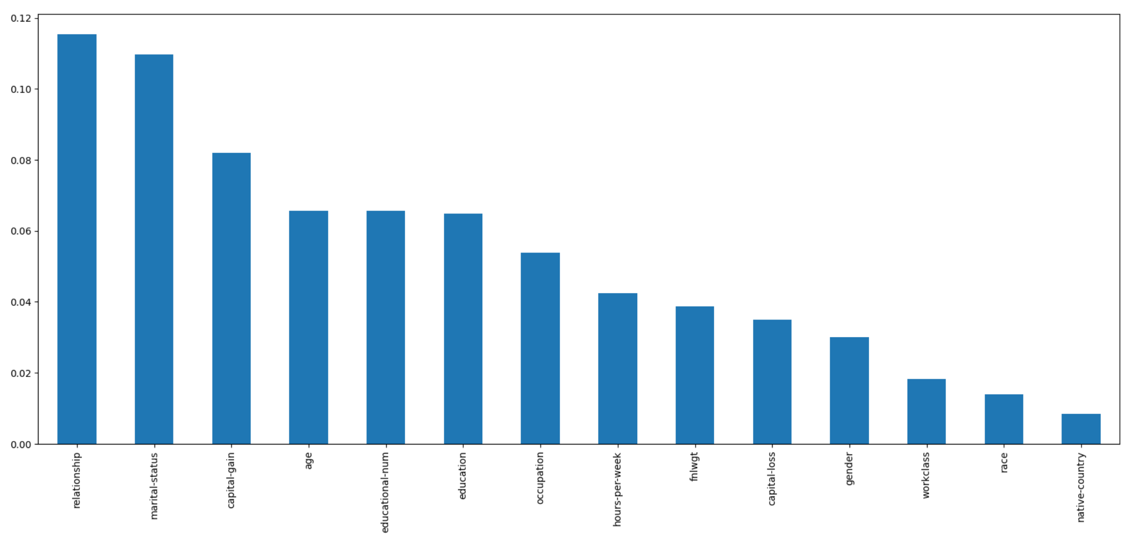
[267]:
```python
mutual_info = pd.Series(mutual_info)
mutual_info.index = X.columns
mutual_info.sort_values(ascending=False)
```

[267]:
```
relationship       0.115342
marital-status     0.109612
```

```
capital-gain       0.082051
age                0.065713
educational-num    0.065588
education          0.064791
occupation         0.053871
hours-per-week     0.042513
fnlwgt             0.038678
capital-loss       0.035040
gender             0.030031
workclass          0.018267
race               0.013916
native-country     0.008430
dtype: float64
```

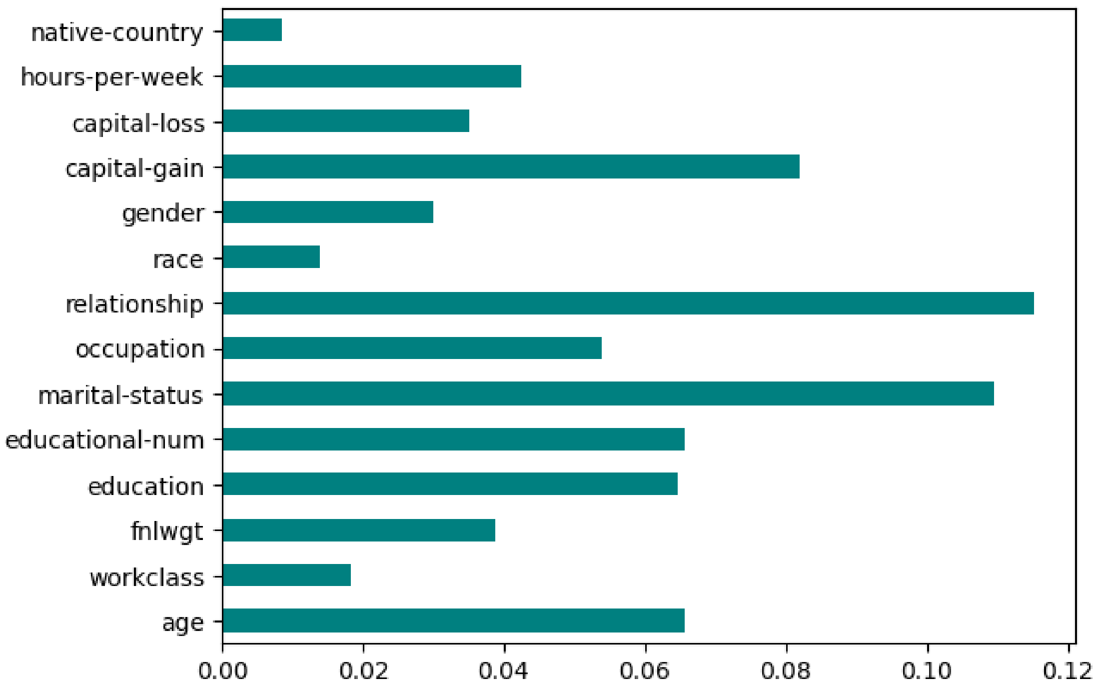
[268]:
```
mutual_info.sort_values(ascending=False).plot.bar(figsize=(20, 8))
```

[268]: <Axes: >



[269]:
```
imp_feat=pd.Series(mutual_info,df.columns[0:len(df.columns)-1])
imp_feat.plot(kind="barh",color="teal")
```

[269]: <Axes: >



[270]:
```
X = df.
↪drop(['workclass','race','native-country','fnlwgt','marital-status','relationship','income']
```

[271]:
```
X
```

[271]:
```
       age  education  educational-num  occupation  gender  capital-gain  \
0       25          5                7           6       1             0
1       38          3                9           4       1             0
2       28          1               12          10       1             0
3       44          1               10           6       1          7688
4       18          1               10           9       0             0
…      …          …                …           …       …             …
48837   27          1               12          12       0             0
48838   40          3                9           6       1             0
48839   58          3                9           0       0             0
48840   22          3                9           0       1             0
48841   52          3                9           3       0         15024

       capital-loss  hours-per-week
0                 0              40
1                 0              50
2                 0              40
3                 0              40
4                 0              30
```

```
…             …            …
48837         0            38
48838         0            40
48839         0            40
48840         0            20
48841         0            40

[48790 rows x 8 columns]
```

[272]:

```
from sklearn.model_selection import train_test_split
x_temp=temp_df.drop('Fruit_Price',axis=1)
y_temp=temp_df['Fruit_Price']
```

[273]:

```
temp_df
```

[273]:

```
   Fruit_Name  Fruit_Price  Fruit_Color_Green  Fruit_Color_Red  \
0           3         1000              False            False
1           0          300              False             True
2           1           50              False            False
3           2          100               True            False

   Fruit_Color_Yellow
0                True
1               False
2                True
3               False
```

[274]:

```
xtrain,xtest,ytrain,ytest=train_test_split(x_temp,y_temp,test_size=0.
↪5,random_state=0,shuffle=True)
```

[275]:

```
print(xtrain.shape, ytrain.shape)
```

```
(2, 4) (2,)
```

[276]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.
↪3,random_state=42,shuffle=True)
```

[277]:

```
print('X_Training Shape: ', X_train.shape)
print('X_Testing Shape: ', X_test.shape)
print('Y_Training Shape: ', y_train.shape)
print('Y_Testing Shape: ', y_test.shape)
```

```
X_Training Shape:  (34153, 8)
X_Testing Shape:  (14637, 8)
Y_Training Shape:  (34153,)
Y_Testing Shape:  (14637,)
```

11 Data Scaling

1. MinMax Scaler
2. Standard Scaler

[278]:
```
from sklearn.preprocessing import StandardScaler

scaler=StandardScaler()

X_train_std=scaler.fit_transform(X_train)
X_test_std=scaler.transform(X_test)
```

[279]:
```
X_train_std.shape
```

[279]:
```
(34153, 8)
```

12 END of Data Preprocessing Assignment

13 Sample Model Development Using Tensorflow (Simple ANN)

[280]:
```
# Model Development
import tensorflow as tf
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam,SGD
```

[281]:
```
model = tf.keras.models.Sequential()
model.add(Dense(8,input_shape=(8,),activation='relu'))
#model.add(Dense(10,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
```

[282]:
```
model.summary()
```

```
Model: "sequential_7"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_16 (Dense)            (None, 8)                 72

 dense_17 (Dense)            (None, 1)                 9

=================================================================
Total params: 81 (324.00 Byte)
Trainable params: 81 (324.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[283]:
```
model.compile(loss='binary_crossentropy', optimizer=SGD(learning_rate=0.
↪001),metrics=['accuracy'])
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs
slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.
keras.optimizers.legacy.SGD`.
```

[284]:
```
history=model.fit(X_train_std,y_train,batch_size=32,epochs=10)
```

```
Epoch 1/10
1068/1068 [==============================] - 1s 360us/step - loss: 0.6948 -
accuracy: 0.5994
Epoch 2/10
1068/1068 [==============================] - 0s 353us/step - loss: 0.5454 -
accuracy: 0.7775
Epoch 3/10
1068/1068 [==============================] - 0s 354us/step - loss: 0.5023 -
accuracy: 0.7804
Epoch 4/10
1068/1068 [==============================] - 0s 353us/step - loss: 0.4815 -
accuracy: 0.7794
Epoch 5/10
1068/1068 [==============================] - 0s 351us/step - loss: 0.4681 -
accuracy: 0.7812
Epoch 6/10
1068/1068 [==============================] - 0s 358us/step - loss: 0.4585 -
accuracy: 0.7844
Epoch 7/10
1068/1068 [==============================] - 0s 353us/step - loss: 0.4515 -
accuracy: 0.7863
Epoch 8/10
1068/1068 [==============================] - 0s 353us/step - loss: 0.4461 -
accuracy: 0.7887
Epoch 9/10
1068/1068 [==============================] - 0s 352us/step - loss: 0.4419 -
accuracy: 0.7902
Epoch 10/10
1068/1068 [==============================] - 0s 353us/step - loss: 0.4384 -
accuracy: 0.7912
```

[287]:
```
from sklearn.metrics import accuracy_score
y_pred = np.argmax(model.predict(X_test),axis=1)
accuracy = accuracy_score(y_test, y_pred)
```

```
458/458 [==============================] - 0s 272us/step [
```

288]:
```
print(accuracy)
```

```
0.7579422012707522
```