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

```
df= pd.read_csv('income.csv', error_bad_lines=False)
```

```
<ipython-input-3-0beac38d255e>:1: FutureWarning: The error_bad_lines argument has been deprecated and will be removed in a future v

  df= pd.read_csv('income.csv', error_bad_lines=False)
```

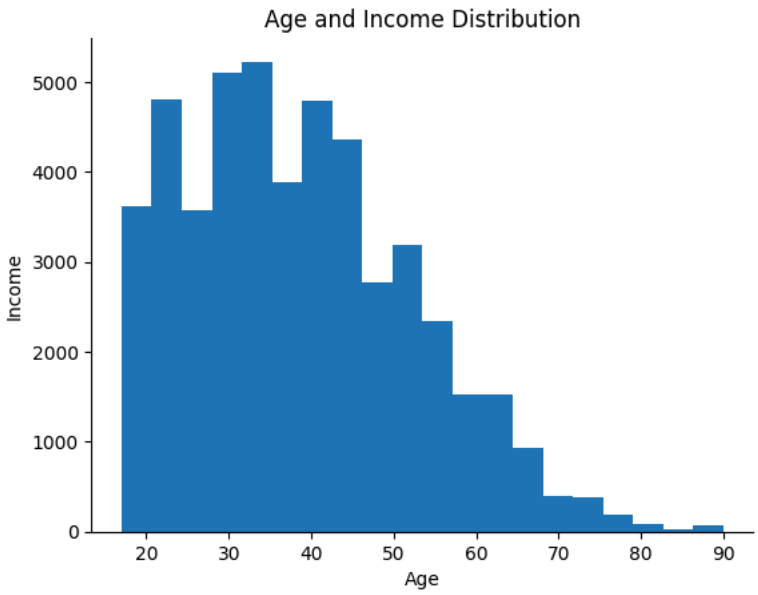
Data Exploration

```
df
```

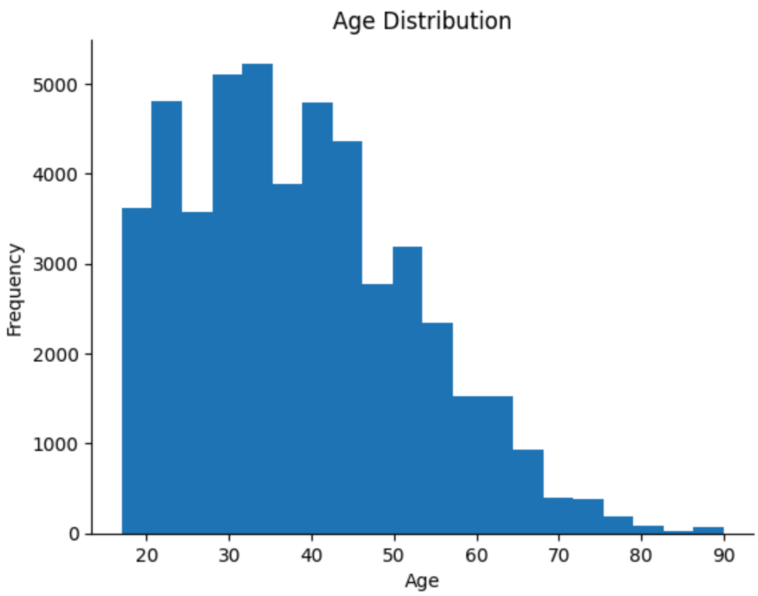
	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relatic
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Ow
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Hu
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Hu
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Hu
4	18	?	103497	Some-college	10	Never-married	?	Ow
...	...	...	...	...	...	...	...	
48837	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	
48838	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Hu
48839	58	Private	151910	HS-grad	9	Widowed	Adm-	Unr

Next steps: View recommended plots

```
## Ages and income
df['age'].plot(kind='hist', bins=20, title='Age and Income Distribution')
plt.xlabel('Age')
plt.ylabel('Income')
plt.gca().spines[['top', 'right']].set_visible(False)
```




```
## age and count
df['age'].plot(kind='hist', bins=20, title='Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.gca().spines[['top', 'right']].set_visible(False)
```



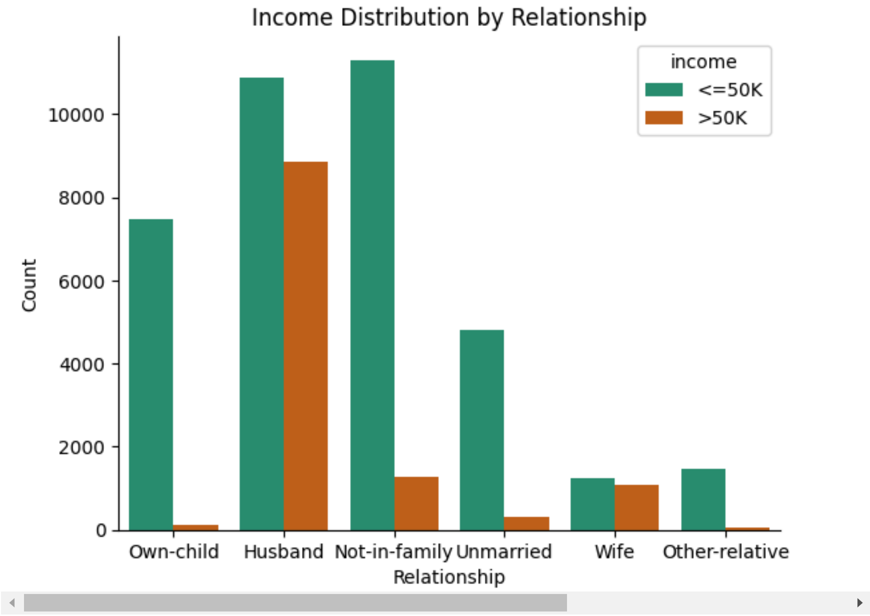
Data Visualisation

```
df.relationship.value_counts()
```

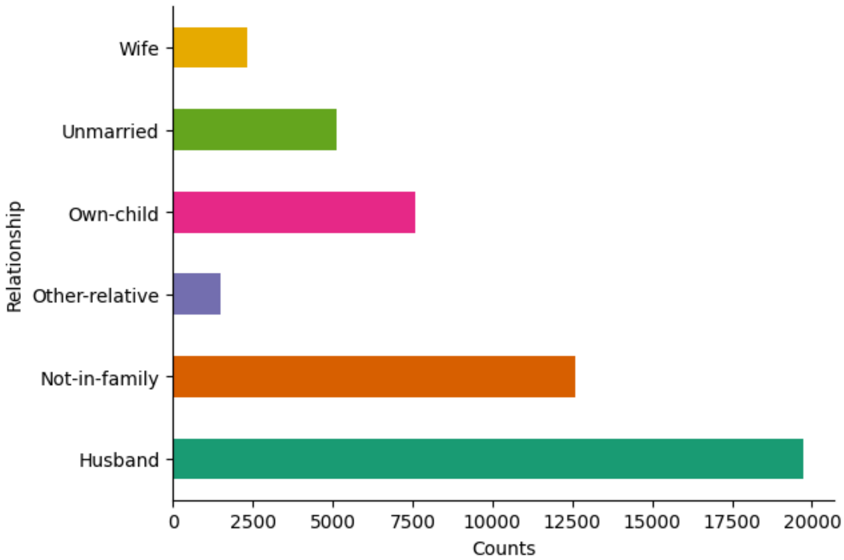
```
Husband           19716
Not-in-family     12583
Own-child          7581
Unmarried          5125
Wife               2331
Other-relative     1506
Name: relationship, dtype: int64
```

```
sns.countplot(data=df, x='relationship', hue='income', palette=sns.palettes.mpl_palette('Dark2'))
plt.xlabel('Relationship')
plt.ylabel('Count')
plt.title('Income Distribution by Relationship')
plt.gca().spines[['top', 'right']].set_visible(False)
plt.show()
```

```
<ipython-input-11-752f33d28d3a>:1: UserWarning: The palette list has more values (6)
  sns.countplot(data=df, x='relationship', hue='income', palette=sns.palettes.mpl_pa
```

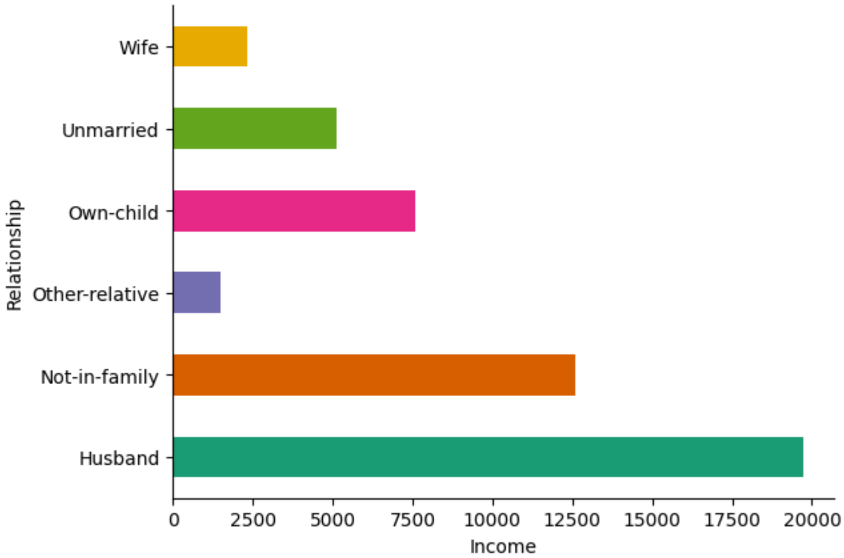



```
## Relationship and counts
df.groupby('relationship').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
plt.xlabel('Counts')
plt.ylabel('Relationship')
plt.gca().spines[['top', 'right']].set_visible(False)
```

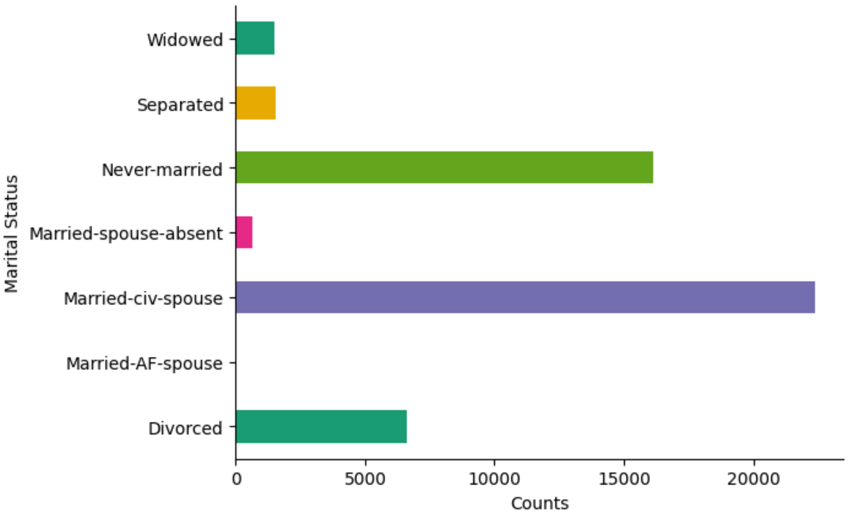



```
## relationship and income

df.groupby('relationship').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
plt.xlabel('Income')
plt.ylabel('Relationship')
plt.gca().spines[['top', 'right']].set_visible(False)
```




```
## Marital stastus
df.groupby('marital-status').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
plt.xlabel('Counts')
plt.ylabel('Marital Status')
plt.gca().spines[['top', 'right']].set_visible(False)
```




```
df.education.value_counts()
```

```
df.workclass.value_counts()
```

```
df.occupation.value_counts()
```

```
## concat the dummies and add prefix
df= pd.concat ([df.drop('occupation', axis=1), pd.get_dummies(df.occupation).add_prefix('occupation_')], axis=1)
```

```
df = pd.concat([df.drop('workclass', axis=1), pd.get_dummies(df['workclass']).add_prefix('workclass_')], axis=1)
df = pd.concat([df.drop('marital-status', axis=1), pd.get_dummies(df['marital-status']).add_prefix('marital_status_')], axis=1)
df = pd.concat([df.drop('relationship', axis=1), pd.get_dummies(df['relationship']).add_prefix('relationship_')], axis=1)
df = pd.concat([df.drop('race', axis=1), pd.get_dummies(df['race']).add_prefix('race_')], axis=1)
df = pd.concat([df.drop('native-country', axis=1), pd.get_dummies(df['native-country']).add_prefix('native_country_')], axis=1)
```

```
df.drop('education', axis=1, inplace=True)
```

```
df
```

```
## Numberical value for income and gender
df['income']= df['income'].apply(lambda x : 1 if x == '>50K' else 0)


df['gender']= df['gender'].apply(lambda x : 1 if x == 'Male' else 0)
```

```
df
```

```
df.columns.values
```

```
## Too Much columns
df.corr()
```

	age	fnlwgt	educational-num	gender	capital-gain
age	1.000000	-0.076628	0.030940	0.088120	0.077229
fnlwgt	-0.076628	1.000000	-0.038761	0.027739	-0.003706
educational-num	0.030940	-0.038761	1.000000	0.009328	0.125146
gender	0.088120	0.027739	0.009328	1.000000	0.047094
capital-gain	0.077229	-0.003706	0.125146	0.047094	1.000000
...	...	...	...	...	...
native_country_Thailand	-0.001766	-0.001512	0.007283	-0.007117	-0.002781
native_country_Trinadad&Tobago	0.001056	0.004153	-0.010201	-0.009342	-0.003039
native_country_United-States	0.011888	-0.070645	0.104210	-0.011167	0.004191
native_country_Vietnam	-0.012337	-0.007479	-0.007544	-0.001545	-0.002673
native_country_Yugoslavia	0.002905	0.004699	-0.005798	0.005262	-0.000474

92 rows × 92 columns

```
## Filter the columns
correlations = df.corr()['income'].abs()
sorted_correlations = correlations.sort_values()
num_cols_to_drop = int(0.8* len(df.columns))
cols_to_drop = sorted_correlations.iloc[:num_cols_to_drop].index
df_dropped= df.drop(cols_to_drop, axis=1)
```

```
df_dropped
```

	age	educational-num	gender	capital-gain	capital-loss	hours-per-week	income	occupation_Exec manageria
0	25	7	1	0	0	40	0	
1	38	9	1	0	0	50	0	
2	28	12	1	0	0	40	1	
3	44	10	1	7688	0	40	1	
4	18	10	0	0	0	30	0	
...	...	...	...	...	...	...	...	
48837	27	12	0	0	0	38	0	
48838	40	9	1	0	0	40	1	
48839	58	9	0	0	0	40	0	
48840	22	9	1	0	0	20	0	
48841	52	9	0	15024	0	40	1	

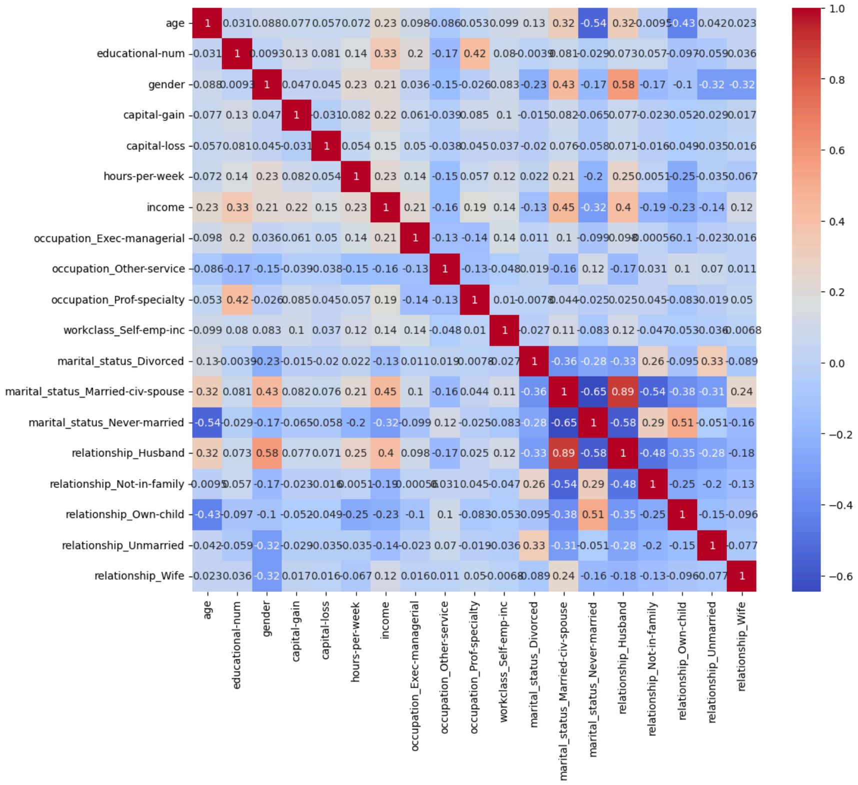
48842 rows × 19 columns

Next steps: View recommended plots

```
plt.figure(figsize=(12,10))
sns.heatmap(df_dropped.corr(), annot=True, cmap='coolwarm')

## Income high correlated with marital_status_Married-civ-spouse
## Income with husband second
## Income with edu number
```

```
<Axes: >
```



2 Machine Learning Model

1)Decision Tree (Factors higher the salaries)

2)Linear Regression (Predic Monthly salaries)

```
## Model (decision tree)

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

## Drop fnlwgt

df= df.drop('fnlwgt', axis= 1)
train_df, test_df = train_test_split (df, test_size= 0.2)
```

```
train_df
```

	age	educational-num	gender	capital-gain	capital-loss	hours-per-week	income	occupation_?	o
33025	51	13	0	0	0	60	0	0	
21137	32	9	0	0	0	50	0	0	
15792	54	10	0	0	0	40	1	0	
48655	23	11	0	0	0	40	0	0	
47838	33	10	1	0	0	45	0	0	
...	...	...	...	...	...	...	...	...	
39280	27	13	1	0	0	35	0	0	
45035	17	7	1	0	0	16	0	0	
5844	43	10	1	0	0	45	0	0	
24296	34	9	1	0	0	55	1	0	
22048	50	10	1	0	0	40	1	0	

39073 rows × 91 columns

```
test_df
```

	age	educational-num	gender	capital-gain	capital-loss	hours-per-week	income	occupation_?	o
31370	38	13	1	0	0	50	1	0	
20140	48	9	0	0	0	40	0	0	
30704	29	9	1	0	0	40	0	0	
100	51	10	0	0	0	18	0	1	
44381	58	10	1	0	0	12	0	0	
...	...	...	...	...	...	...	...	...	
28613	40	4	1	0	0	40	0	0	
3304	55	10	1	0	0	40	0	0	
28608	40	12	1	0	0	52	0	0	
32333	51	9	1	0	0	50	1	0	
45720	32	9	0	0	0	40	0	0	

9769 rows × 91 columns

```
train_X= train_df.drop('income', axis=1)
train_Y= train_df['income']

test_X= test_df.drop('income', axis=1)
test_Y= test_df['income']
```

```
forest = RandomForestClassifier()
forest.fit(train_X, train_Y)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
## Test the scores

## 0.85 percent is quite good
forest.score(test_X,test_Y)
```

```
0.8583273620636708
```

```
forest.feature_importances_
```

```
forest.feature_names_in_
```

```
importances =dict(zip(forest.feature_names_in_, forest.feature_importances_))
importances = {k: v for k, v in sorted(importances.items(), key= lambda x : x[1], reverse= True)}
```

```
importances

## Conclusion
## The older you get the higher the income
## The higher the education number the higher the income
## The more you work per week the higher your income
```

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [50,100,250],
    'max_depth': [5,10,30,None],
    'min_samples_split': [2,4],
    'max_features': ['sqrt','log2']
}

grid_search = GridSearchCV(estimator=RandomForestClassifier(),
                           param_grid=param_grid, verbose=10)
```

```
grid_search.fit(train_X, train_Y)
```

```
grid_search.best_estimator_
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=30, min_samples_split=4, n_estimators=250)
```

```
forest.score(test_X,test_Y)
```

```
0.8583273620636708
```

```
### Second Model Development using LinearRegression

from sklearn.linear_model import LinearRegression
Linear_Regression_model = LinearRegression
from sklearn.metrics import mean_squared_error
```

```
## Dependent and independent variables
## Spilt the data into training and testing

x = df[['age', 'gender']]
y = df['income']

from sklearn.model_selection import train_test_split

train_X, test_X, train_Y, test_Y = train_test_split (x,y, test_size=0.2, random_state=42)

train_X.head()
```

	age	gender
37193	42	1
31093	52	1
33814	34	1
14500	28	0
23399	46	1

Next steps: View recommended plots

```
df
```

	age	educational-num	gender	capital-gain	capital-loss	hours-per-week	income	occupation_?	c
0	25	7	1	0	0	40	0	0	
1	38	9	1	0	0	50	0	0	
2	28	12	1	0	0	40	1	0	
3	44	10	1	7688	0	40	1	0	
4	18	10	0	0	0	30	0	1	
...	...	...	...	...	...	...	...	...	
48837	27	12	0	0	0	38	0	0	
48838	40	9	1	0	0	40	1	0	
48839	58	9	0	0	0	40	0	0	
48840	22	9	1	0	0	20	0	0	
48841	52	9	0	15024	0	40	1	0	

48842 rows × 91 columns

```
train_X.shape, train_Y.shape,
```

```
((39073, 2), (39073,))
```

```
test_X.shape, test_Y.shape
```

```
((9769, 2), (9769,))
```

```
## Model Training

model = LinearRegression()
model.fit(train_X, train_Y)
```

```
▾ LinearRegression
LinearRegression()
```

```
## Model Prediction

y_pred = model.predict(test_X)
y_pred
```

```
array([0.23936549, 0.20728376, 0.1519606 , ..., 0.03093846, 0.16021959,
       0.38881698])
```

```
mse = mean_squared_error(test_Y, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.16354416612069345
```

Make Prediction

Employees Detailst

Name = Ali, Aminah, Abu

Employees No = 1,2,3

Age= 30,40,50

Gender = 'Male'= 1, 'Female'= 0

Ringgit Per Minutes

```
Monthly_salary = pd.DataFrame({'age': [30, 40, 50], 'gender': ['1', '0', '1']})

# Make predictions on monthly salary
situations = model.predict(Monthly_salary)

# Print the predictions
print("Predictions for employees:")
for i, prediction in enumerate(situations):
    print(f"Prediction for employees {i+1}: {prediction}")
```

```
Predictions for employees:
Prediction for employees 1: 0.24090102270624994
Prediction for employees 2: 0.13179024607072595
Prediction for employees 3: 0.3753700772692956
```

```
###Ringgit Malaysia Per Minutes

Monthly_salary = pd.DataFrame({'age': [30, 40, 50], 'gender': ['1', '0', '1']})

# Make predictions on monthly salary
situations = model.predict(Monthly_salary)

# Print the predictions in Ringgit Malaysia
print("Predictions for employees: (in Ringgit Malaysia):")
for i, prediction in enumerate(situations):
    print(f"Prediction for employees {i+1}: RM {prediction:.2f}")
```

```
Predictions for employees: (in Ringgit Malaysia):
Prediction for employees 1: RM 0.24
Prediction for employees 2: RM 0.13
Prediction for employees 3: RM 0.38
```

```
### Employees 1 (RM0.24*60*8*30) = RM3456 Monthly as he working 8 hours perday and 30 days monthly. Male gender
### Employees 2 (RM0.13*60*8*30) = RM1872 Monthly as he working 8 hours perday and 30 days monthly. Lower rate because Female
### Employees 3 (RM0.38*60*8*30) = RM5472 Monthly as he working 8 hours perday and 30 days monthly. Male gender
```

2 Perfomance Metrix

1) Evaluating Model Performance: Accuracy in Decision Tree for Salary Prediction

In the realm of machine learning, assessing model performance is paramount to ensure reliable predictions. One crucial metric, **accuracy**, measures the proportion of correctly classified instances out of the total instances. In our classification task, centered on predicting factors leading to higher salaries using Decision Tree algorithm, we achieved an accuracy score of **0.850**. This implies that our model accurately predicts the outcome in **85% of cases**, exhibiting commendable performance.

2) Mean Squared Error in Linear Regression for Monthly Salary Prediction

In the domain of predictive modeling, evaluating model performance is crucial to gauge its effectiveness. One key metric, **mean squared error (MSE),** measures the average squared difference between the actual and predicted values. In our task of predicting monthly salaries using Linear Regression, our model achieved an MSE of 0.1635. This implies that, on average, the squared difference between our **predicted salaries and the actual salaries is 0.1635**. While a lower MSE indicates better model performance, it's essential to consider other metrics like R-squared and mean absolute error to gain a holistic understanding. Thus, while acknowledging our accomplishment, let's continue refining our model to enhance its predictive accuracy for monthly salary estimation