```
import warnings
warnings.filterwarnings('ignore')

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

from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.decomposition import TruncatedSVD
from sklearn.pipeline import Pipeline

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClas
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import RocCurveDisplay, roc_curve, auc, classification_report, confusion
```

Load dataset

```
project_data = pd.read_csv("/content/car_data.csv")
```

```
project_data.head()
```

	User ID	Gender	Age	AnnualSalary	Purchased
0	385	Male	35	20000	0
1	681	Male	40	43500	0
2	353	Male	49	74000	0
3	895	Male	40	107500	1
4	661	Male	25	79000	0

```
project_data.tail()
```

	User ID	Gender	Age	AnnualSalary	Purchased
995	863	Male	38	59000	0
997	407	Female	28	138500	1

Exploratory data analysis

```
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   User ID       1000 non-null   int64
 1   Gender        1000 non-null   object
 2   Age           1000 non-null   int64
 3   AnnualSalary  1000 non-null   int64
 4   Purchased     1000 non-null   int64
dtypes: int64(4), object(1)
memory usage: 39.2+ KB
```

```
project_data.describe()
```

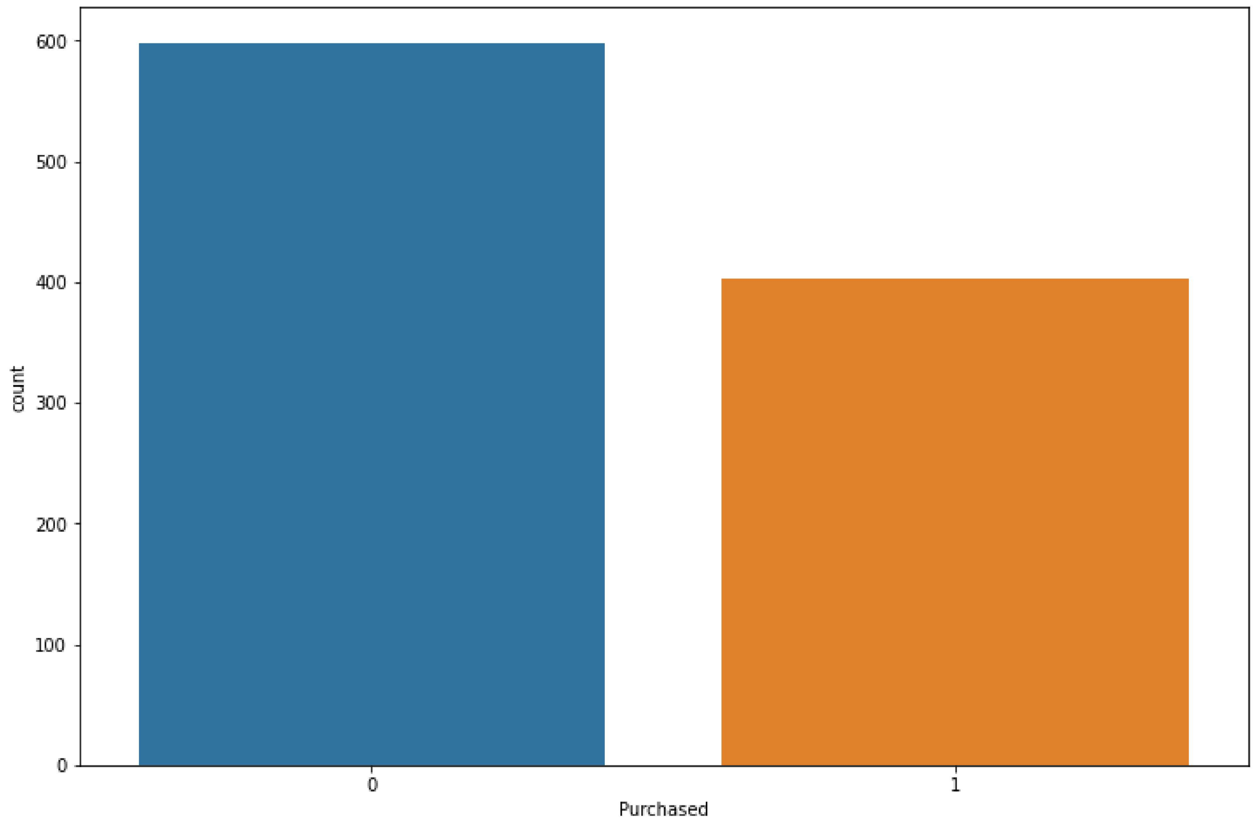
	User ID	Age	AnnualSalary	Purchased
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	500.500000	40.106000	72689.000000	0.402000
std	288.819436	10.707073	34488.341867	0.490547
min	1.000000	18.000000	15000.000000	0.000000
25%	250.750000	32.000000	46375.000000	0.000000
50%	500.500000	40.000000	72000.000000	0.000000
75%	750.250000	48.000000	90000.000000	1.000000
max	1000.000000	63.000000	152500.000000	1.000000

```
project_data.isna().sum()
```

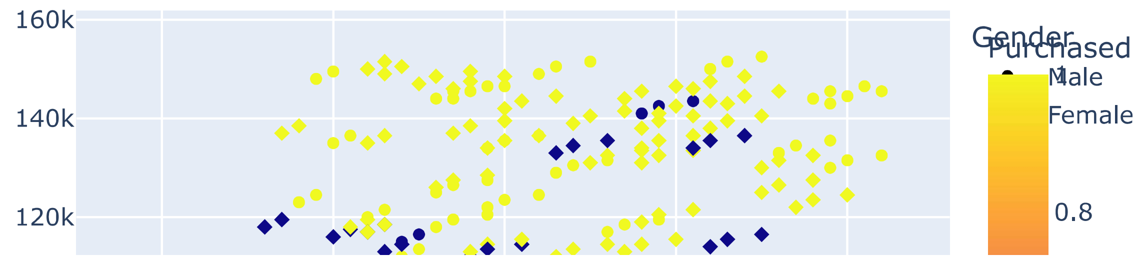
```
User ID         0
Gender          0
Age             0
AnnualSalary    0
Purchased       0
dtype: int64
```

```
plt.figure(figsize=(12,8))
sns.countplot(x='Purchased',data=project_data)
```

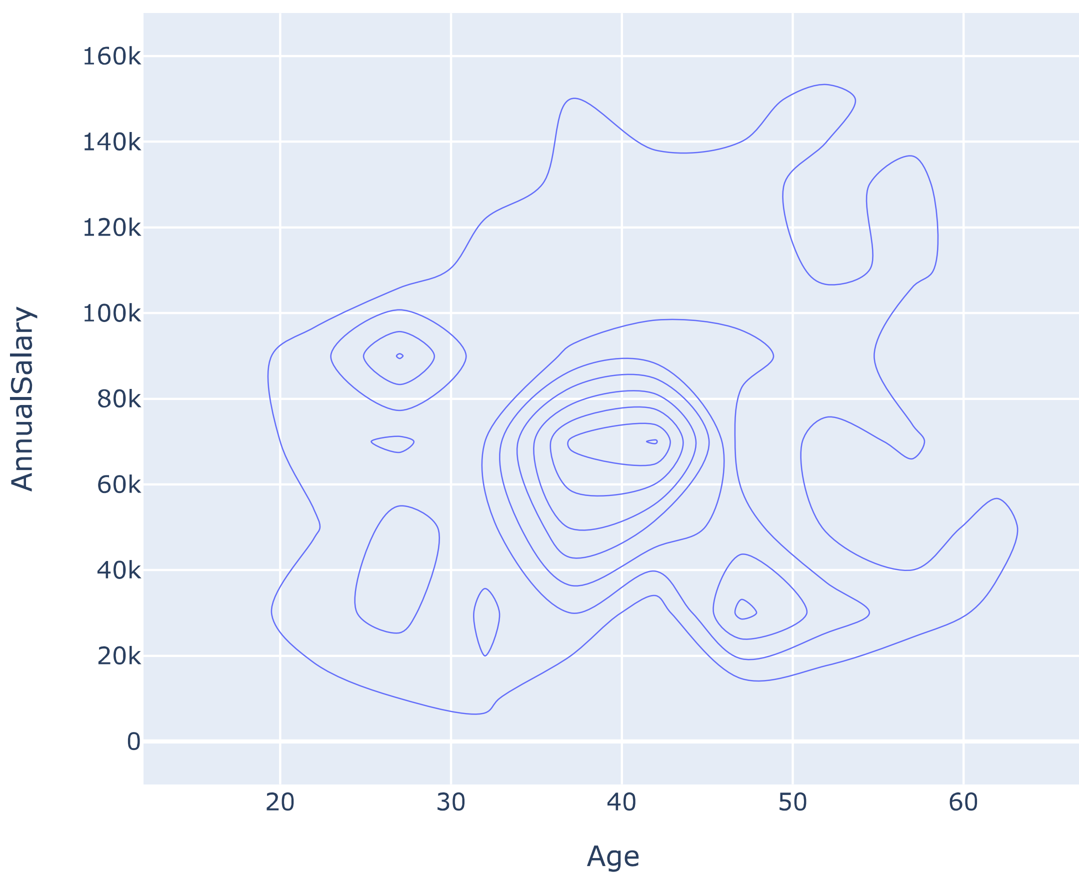
```
plt.show()
```




```
import plotly.express as px
fig = px.scatter(project_data, x='Age', y='AnnualSalary',color='Purchased',symbol='Gender')
fig.show()
```

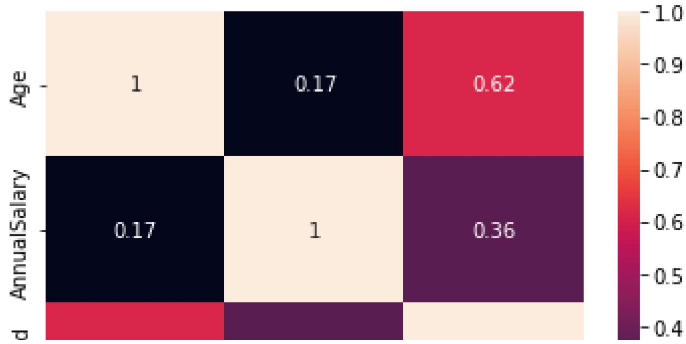



```
fig = px.density_contour(project_data, x='Age', y='AnnualSalary')
fig.show()
```




```
project_data = project_data.drop(columns=['User ID'], axis=1)
corr = project_data.corr()
sns.heatmap(corr, annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5af88091d0>
```



Training the model


```
X_train=project_data.drop(columns="Purchased")
y_train=project_data["Purchased"]

X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.3)


print('Train dataset shape:',X_train.shape)
print('Test dataset shape', y_train.shape)
```

```
Train dataset shape: (700, 3)
Test dataset shape (700,)
```

Data preprocessing

```
numeric_columns = X_train.select_dtypes(exclude='object').columns
print(numeric_columns)

categorical_columns = X_train.select_dtypes(include='object').columns
print(categorical_columns)
```

```
Index(['Age', 'AnnualSalary'], dtype='object')
Index(['Gender'], dtype='object')
```

```
numeric_features = Pipeline([
    ('handlingmissingvalues',SimpleImputer(strategy='median')),
    ('scaling',StandardScaler(with_mean=True))
])

print(numeric_features)
print('*'*100)

categorical_features = Pipeline([
    ('handlingmissingvalues',SimpleImputer(strategy='most_frequent')),
    ('encoding', OneHotEncoder()),
    ('scaling', StandardScaler(with_mean=False))
])
```

```
print(categorical_features)

processing = ColumnTransformer([
    ('numeric', numeric_features, numeric_columns),
    ('categorical', categorical_features, categorical_columns)
])
```

```
Pipeline(steps=[('handlingmissingvalues', SimpleImputer(strategy='median')),
                ('scaling', StandardScaler())])
*************************************************************************************
Pipeline(steps=[('handlingmissingvalues',
                 SimpleImputer(strategy='most_frequent')),
                ('encoding', OneHotEncoder()),
                ('scaling', StandardScaler(with_mean=False))])
```

```
def prepare_model(algorithm):
    model = Pipeline(steps= [
        ('processing',processing),
        ('pca', TruncatedSVD(n_components=3, random_state=12)),
        ('modeling', algorithm)
    ])
    model.fit(X_train, y_train)
    return model


def prepare_classification_report(algo, model):
    print(algo+' Report :')
    pred = model.predict(X_test)
    print(classification_report(y_test, pred))


def prepare_roc_curve(algo, model):
    print(algo)
    y_pred_proba = model.predict_proba(X_test)[::,1]
    fpr, tpr, thresholds = roc_curve(y_test,  y_pred_proba)
    roc_auc = auc(fpr, tpr)
    curve = RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc)
    curve.plot()
    plt.show()
```

Evaluating the following models:

1. Logistic regression
2. KNN classifier
3. Random forest classifier
4. Adaboost classifier
5. Gradientboost classifier

```
algorithms = [('logistic regression', LogisticRegression()),
              ('KNN classifier', KNeighborsClassifier()),
              ('Random Forest calssifier', RandomForestClassifier()),
              ('Adaboost classifier', AdaBoostClassifier()),
              ('Gradientboot classifier',GradientBoostingClassifier())
             ]

trained_models = []
model_and_score = {}

for index, tup in enumerate(algorithms):
    model = prepare_model(tup[1])
    model_and_score[tup[0]] = str(model.score(X_train,y_train)*100)+"%"
    trained_models.append((tup[0],model))
    #prepare_classification_report(tup[0], model)

print(model_and_score)
```

```
{'logistic regression': '81.85714285714286%', 'KNN classifier': '92.42857142857143%', 'R
```

We get 99.7 % accuracy with random forest classifier

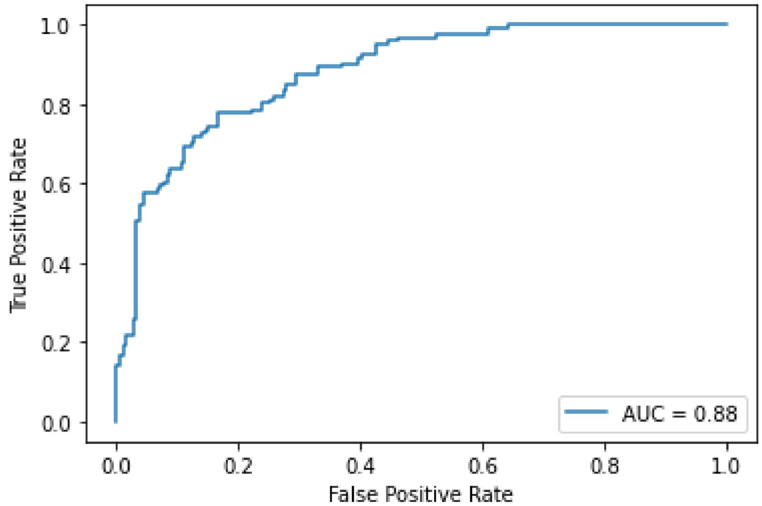
```
for index, tup in enumerate(trained_models):
    prepare_classification_report(tup[0], tup[1])
    print("\n")
```

```
logistic regression Report :
              precision    recall  f1-score   support

           0       0.82      0.86      0.84       179
           1       0.78      0.72      0.75       121

    accuracy                           0.80       300
   macro avg       0.80      0.79      0.79       300
weighted avg       0.80      0.80      0.80       300



KNN classifier Report :
              precision    recall  f1-score   support

           0       0.90      0.88      0.89       179
           1       0.83      0.85      0.84       121

    accuracy                           0.87       300
   macro avg       0.86      0.87      0.87       300
weighted avg       0.87      0.87      0.87       300
```

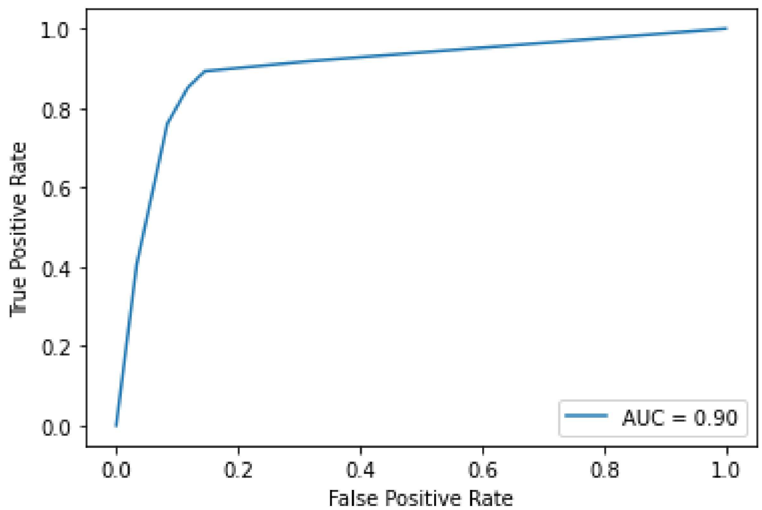
```
Random Forest calssifier Report :
              precision    recall  f1-score   support

           0       0.90      0.86      0.88       179
           1       0.81      0.86      0.83       121

    accuracy                           0.86       300
   macro avg       0.85      0.86      0.86       300
weighted avg       0.86      0.86      0.86       300



Adaboost classifier Report :
              precision    recall  f1-score   support

           0       0.84      0.88      0.86       179
           1       0.81      0.76      0.79       121

    accuracy                           0.83       300
   macro avg       0.83      0.82      0.82       300
weighted avg       0.83      0.83      0.83       300



Gradientboot classifier Report :
              precision    recall  f1-score   support

           0       0.90      0.88      0.89       179
           1       0.83      0.86      0.85       121

    accuracy                           0.87       300
   macro avg       0.87      0.87      0.87       300
weighted avg       0.87      0.87      0.87       300
```

```
for index, tup in enumerate(trained_models):
    prepare_roc_curve(tup[0], tup[1])
```

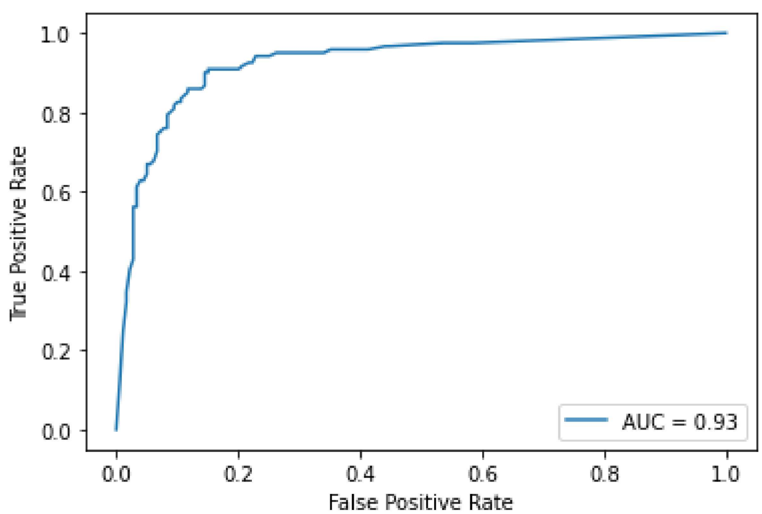
logistic regression



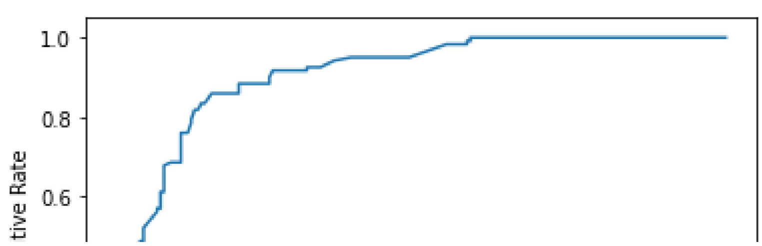
KNN classifier



Random Forest calssifier



Adaboost classifier




Posi