Assignment_Week_9

January 20, 2021

[204]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
```

[205]:
```
df = pd.read_csv("Data01.csv")
```

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

[206]:
```
(400, 5)
```

[66]:
```
df.head()
```

[66]:
```
    User ID  Gender   Age EstimatedSalary  Purchased
0  15624510       M   $19           19000          0
1  15810944    Male    35           20000          0
2  15668575  Female    26          43000%          0
3  15603246  Female    27           57000          0
4  15804002    Male   NaN           76000          0
```

[67]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   User ID          400 non-null    int64
 1   Gender           400 non-null    object
 2   Age              394 non-null    object
 3   EstimatedSalary  395 non-null    object
 4   Purchased        400 non-null    int64
dtypes: int64(2), object(3)
memory usage: 15.8+ KB
```

[68]:
```
df['Gender'].value_counts()
```

[68]:
```
Female     201
Male       191
M            4
F            2
Female%      1
Male%        1
Name: Gender, dtype: int64
```

Clean up data series df[‘Gender’] using “replace” and “map” in dataframe

[207]:
```
new_Gender = { "M":"Male", "F":"Female", "Female%":"Female", "Male%":"Male" }
df = df.replace({"Gender":new_Gender})
```

[208]:
```
df['Gender'].value_counts()
```

[208]:
```
Female    204
Male      196
Name: Gender, dtype: int64
```

[209]:
```
df['Gender'].unique()
```

[209]:
```
array(['Male', 'Female'], dtype=object)
```

[210]:
```
df['Purchased'].unique()
```

[210]:
```
array([0, 1])
```

Clean up $ in df[‘Age’]

[211]:
```
df['Age'].unique()
```

[211]:
```
array(['$19 ', '35', '26', '27', nan, '32', '25', '20', '18', '29', '47',
       '45', '46', '48', '49', '31', '21', '28', '33', '30', '23', '24',
       '22', '59', '34', '39', '19', '38', '37', '42', '40', '36', '41',
       '58', '55', '52', '60', '53', '50', '56', '51', '57', '44', '43',
       '54'], dtype=object)
```

[212]:
```
df['Age'] = df['Age'].str.replace('$','')
```

[213]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.NaN, strategy='mean')
df['Age'] = imputer.fit_transform(df['Age'].values.reshape(-1,1))
```

[214]:
```
df['Age'].unique()
```

[214]:
```
array([19.        , 35.        , 26.        , 27.        , 37.58629442,
       32.        , 25.        , 20.        , 18.        , 29.        ,
       47.        , 45.        , 46.        , 48.        , 49.        ,
```

```
       31.          , 21.          , 28.          , 33.          , 30.          ,
       23.          , 24.          , 22.          , 59.          , 34.          ,
       39.          , 38.          , 37.          , 42.          , 40.          ,
       36.          , 41.          , 58.          , 55.          , 52.          ,
       60.          , 53.          , 50.          , 56.          , 51.          ,
       57.          , 44.          , 43.          , 54.          ])
```

[215]:
```
df['Age'] = df['Age'].astype('int64')
```

[216]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   User ID          400 non-null    int64
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64
 3   EstimatedSalary  395 non-null    object
 4   Purchased        400 non-null    int64
dtypes: int64(3), object(2)
memory usage: 15.8+ KB
```

[217]:
```
df['Age'].unique()
```

[217]:
```
array([19, 35, 26, 27, 37, 32, 25, 20, 18, 29, 47, 45, 46, 48, 49, 31, 21,
       28, 33, 30, 23, 24, 22, 59, 34, 39, 38, 42, 40, 36, 41, 58, 55, 52,
       60, 53, 50, 56, 51, 57, 44, 43, 54])
```

[218]:
```
df['EstimatedSalary'].unique()
```

[218]:
```
array(['19000', '20000', '43000%', '57000', '76000', '58000', '84000',
       '150000', '33000', '65000', '$80,000 ', '52000', '86000', '18000',
       '82000', '80000', '25000', '26000', '28000', '29000', '22000',
       '49000', '41000', '23000', '30000', '43000', '$18,000 ', '74000',
       '137000', '16000', '44000', '90000', '27000', '72000', '31000',
       '17000', '51000', '108000', '15000', '79000', '54000', '135000',
       '89000', '32000', '83000', '55000', '48000', '117000', '87000',
       '66000', '120000', '63000', '68000', '113000', '112000', '42000',
       '88000', '62000', '118000', '85000', '81000', '50000', '116000',
       '$15,000 ', '123000', '73000', '37000', '59000', '149000', '21000',
       '35000', '71000', '61000', '75000', '53000', '107000', '96000',
       '45000', '47000', '100000', '38000', '69000', '148000', '115000',
       '34000', '60000', '70000', '36000', '39000', '134000', '101000',
       '130000', '114000', '142000', nan, '78000', '143000', '91000',
       '144000', '102000', '126000', '133000', '147000', '104000',
```

```
         '146000', '$122,000 ', '97000', '95000', '131000', '77000',
         '125000', '106000', '141000', '93000', '138000', '119000',
         '122000', '105000', '99000', '129000', '46000', '64000', '139000'],
        dtype=object)
```

[219]:
```
df['EstimatedSalary']=df['EstimatedSalary'].str.replace("$","")
df['EstimatedSalary']=df['EstimatedSalary'].str.replace(",","")
df['EstimatedSalary']=df['EstimatedSalary'].str.replace("%","")
```

[220]:
```
imputer = SimpleImputer(missing_values=np.NaN, strategy='mean')
df['EstimatedSalary'] = imputer.fit_transform(df['EstimatedSalary'].values.
 ↪reshape(-1,1))
```

[221]:
```
df['EstimatedSalary'].unique()
```

[221]:
```
array([ 19000.        ,  20000.        ,  43000.        ,  57000.        ,
        76000.        ,  58000.        ,  84000.        , 150000.        ,
        33000.        ,  65000.        ,  80000.        ,  52000.        ,
        86000.        ,  18000.        ,  82000.        ,  25000.        ,
        26000.        ,  28000.        ,  29000.        ,  22000.        ,
        49000.        ,  41000.        ,  23000.        ,  30000.        ,
        74000.        , 137000.        ,  16000.        ,  44000.        ,
        90000.        ,  27000.        ,  72000.        ,  31000.        ,
        17000.        ,  51000.        , 108000.        ,  15000.        ,
        79000.        ,  54000.        , 135000.        ,  89000.        ,
        32000.        ,  83000.        ,  55000.        ,  48000.        ,
       117000.        ,  87000.        ,  66000.        , 120000.        ,
        63000.        ,  68000.        , 113000.        , 112000.        ,
        42000.        ,  88000.        ,  62000.        , 118000.        ,
        85000.        ,  81000.        ,  50000.        , 116000.        ,
       123000.        ,  73000.        ,  37000.        ,  59000.        ,
       149000.        ,  21000.        ,  35000.        ,  71000.        ,
        61000.        ,  75000.        ,  53000.        , 107000.        ,
        96000.        ,  45000.        ,  47000.        , 100000.        ,
        38000.        ,  69000.        , 148000.        , 115000.        ,
        34000.        ,  60000.        ,  70000.        ,  36000.        ,
        39000.        , 134000.        , 101000.        , 130000.        ,
       114000.        , 142000.        ,  70017.72151899,  78000.        ,
       143000.        ,  91000.        , 144000.        , 102000.        ,
       126000.        , 133000.        , 147000.        , 104000.        ,
       146000.        , 122000.        ,  97000.        ,  95000.        ,
       131000.        ,  77000.        , 125000.        , 106000.        ,
       141000.        ,  93000.        , 138000.        , 119000.        ,
       105000.        ,  99000.        , 129000.        ,  46000.        ,
        64000.        , 139000.        ])
```

[222]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   User ID          400 non-null    int64
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64
 3   EstimatedSalary  400 non-null    float64
 4   Purchased        400 non-null    int64
dtypes: float64(1), int64(3), object(1)
memory usage: 15.8+ KB
```

[223]:
```python
df.describe()
```

[223]:

	User ID	Age	EstimatedSalary	Purchased
count	4.000000e+02	400.000000	400.000000	400.000000
mean	1.569154e+07	37.577500	70017.721519	0.357500
std	7.165832e+04	10.336882	33977.615953	0.479864
min	1.556669e+07	18.000000	15000.000000	0.000000
25%	1.562676e+07	30.000000	44000.000000	0.000000
50%	1.569434e+07	37.000000	70017.721519	0.000000
75%	1.575036e+07	45.250000	88000.000000	1.000000
max	1.581524e+07	60.000000	150000.000000	1.000000

[231]:
```python
df.columns
```

[231]:
```
Index(['User ID', 'Gender', 'Age', 'EstimatedSalary', 'Purchased'],
dtype='object')
```

[232]:
```python
df = df.rename(columns={'Age': 'age'})
```

Split data into X the predictors dataframe and y the target series

[233]:
```python
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

[234]:
```python
print(X.shape)
print(y.shape)
```

```
(400, 4)
(400,)
```

[235]:
```python
print(type(X))
print(type(y))
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
```

[236]:
```
X.head()
```

[236]:
```
    User ID  Gender  age  EstimatedSalary
0  15624510    Male   19          19000.0
1  15810944    Male   35          20000.0
2  15668575  Female   26          43000.0
3  15603246  Female   27          57000.0
4  15804002    Male   37          76000.0
```

[237]:
```
y.head()
```

[237]:
```
0    0
1    0
2    0
3    0
4    0
Name: Purchased, dtype: int64
```

Find the correlation between the three predictors [‘Age’, ‘EstimatedSalary’, ‘Gender’] and the target ‘Purchaesd’

[238]:
```
corr1 = np.corrcoef(df['age'], df['Purchased'])
corr1
```

[238]:
```
array([[1.        , 0.61158189],
       [0.61158189, 1.        ]])
```

[239]:
```
corr2 = np.corrcoef(df['EstimatedSalary'], df['Purchased'])
corr2
```

[239]:
```
array([[1.        , 0.36684361],
       [0.36684361, 1.        ]])
```

How about the correlation between two numerical predictors ‘Age’ and ‘Estimated-Salary’

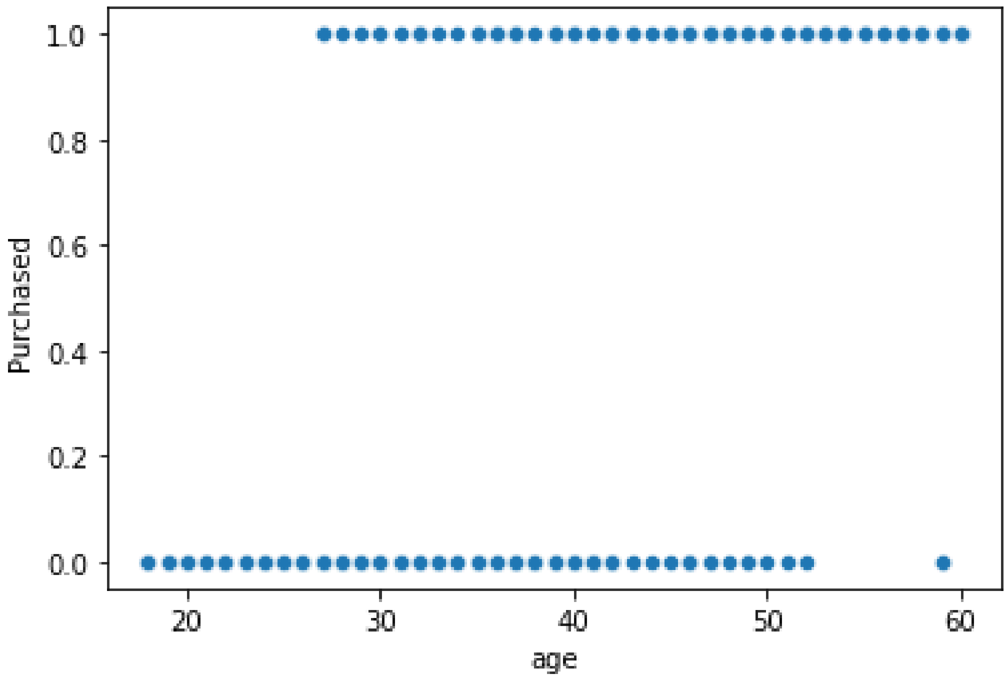
We want to check whether “multicollinearity” is a problem:

[241]:
```
corr2 = np.corrcoef(df['EstimatedSalary'], df['age'])
corr2
```

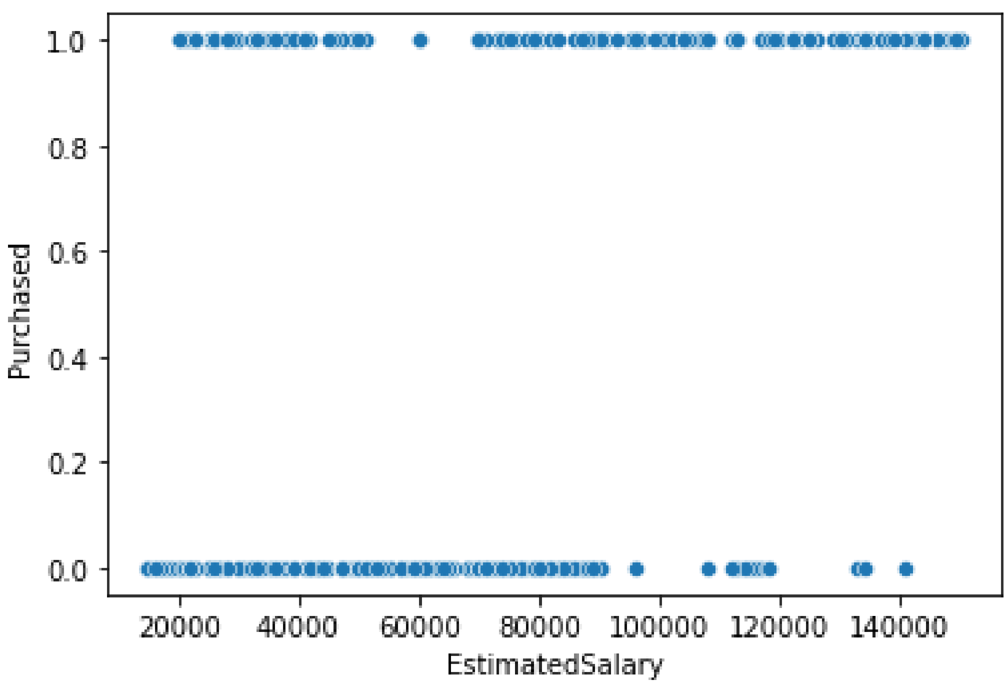
[241]:
```
array([[1.       , 0.1694131],
       [0.1694131, 1.       ]])
```

EDA using Visualization

[242]:
```
sns.scatterplot(data=df, x="age", y="Purchased")
plt.show()
```

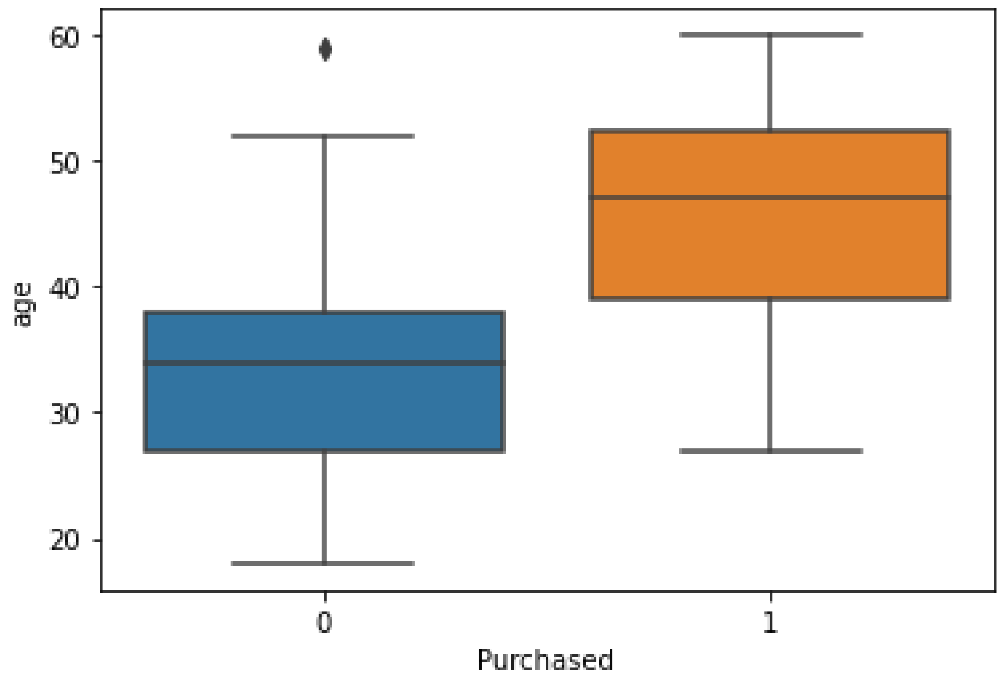


[243]:
```
sns.scatterplot(data=df, x="EstimatedSalary", y="Purchased")
plt.show()
```



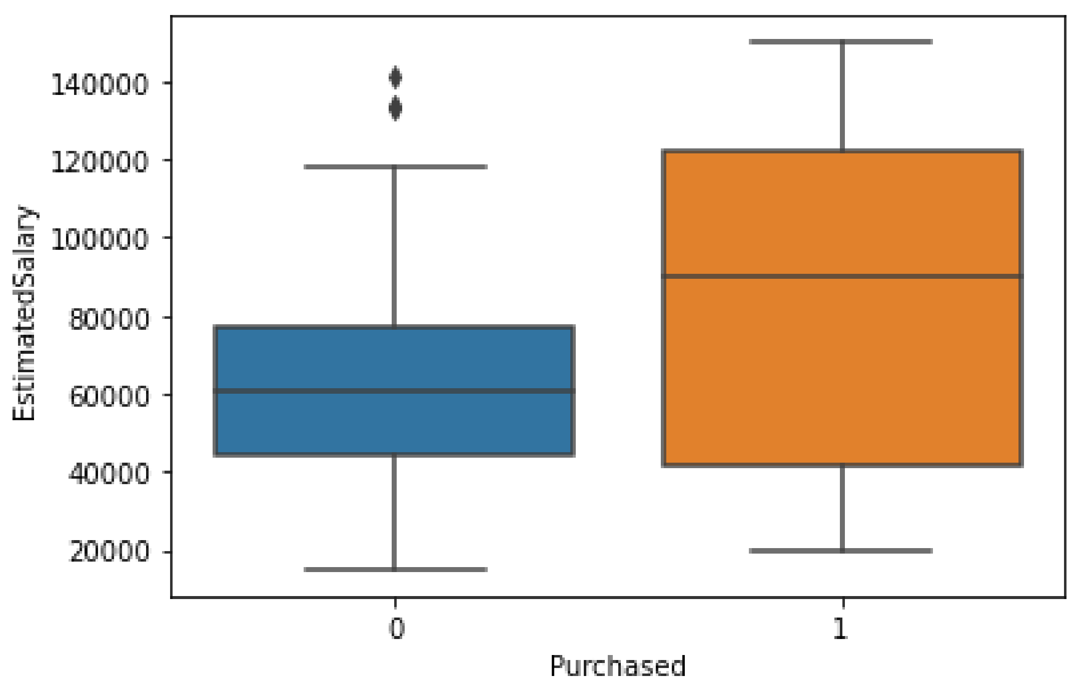
[244]:

```
ax = sns.boxplot(x="Purchased", y="age", data=df)
```

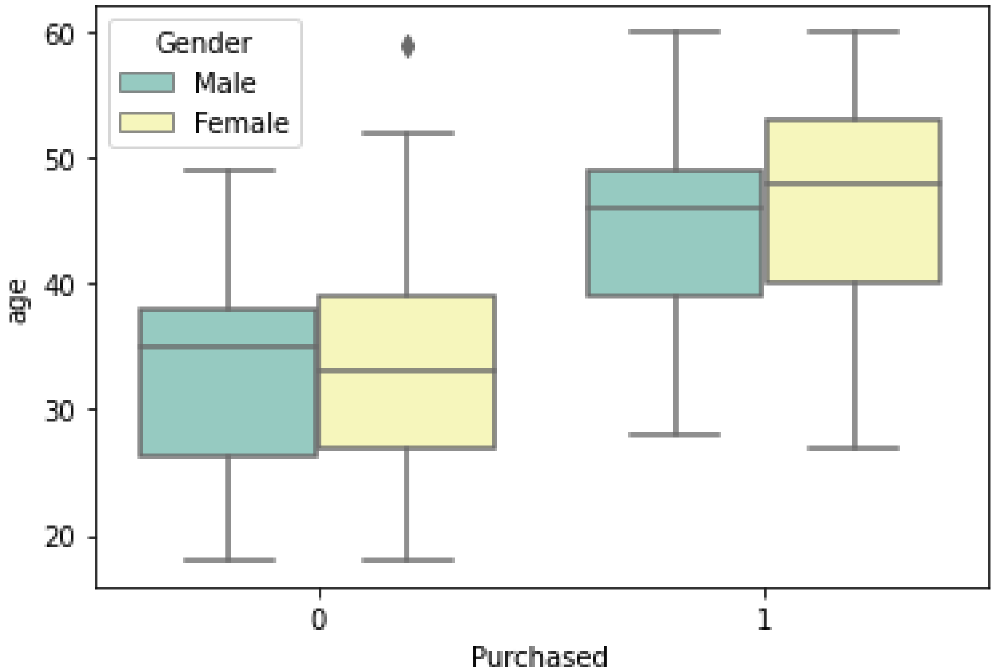


[245]:

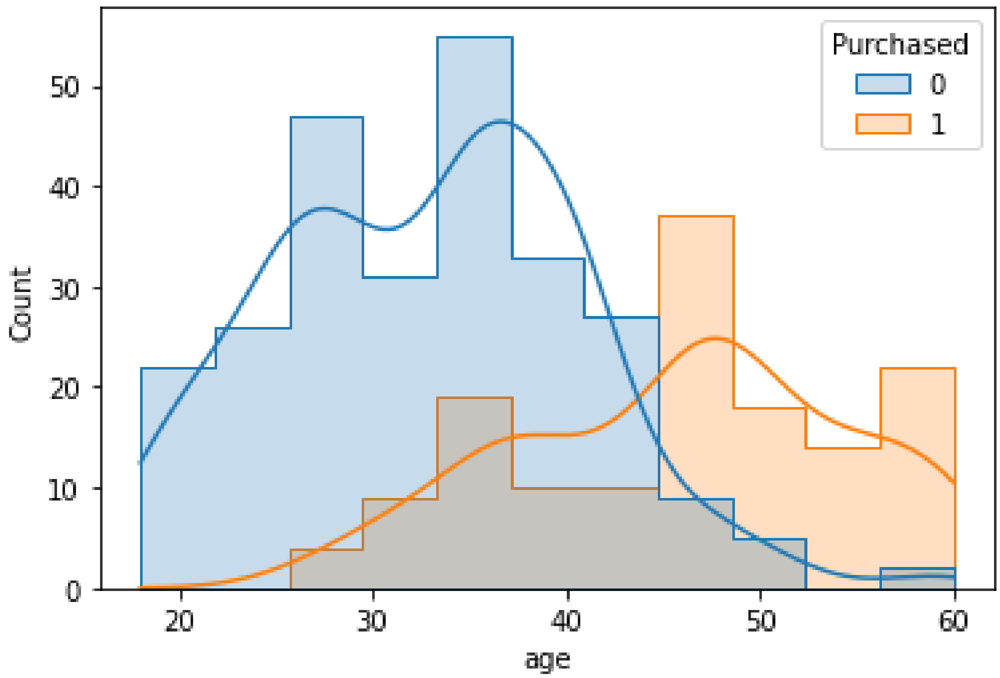
```
ax = sns.boxplot(x="Purchased", y="EstimatedSalary", data=df)
```



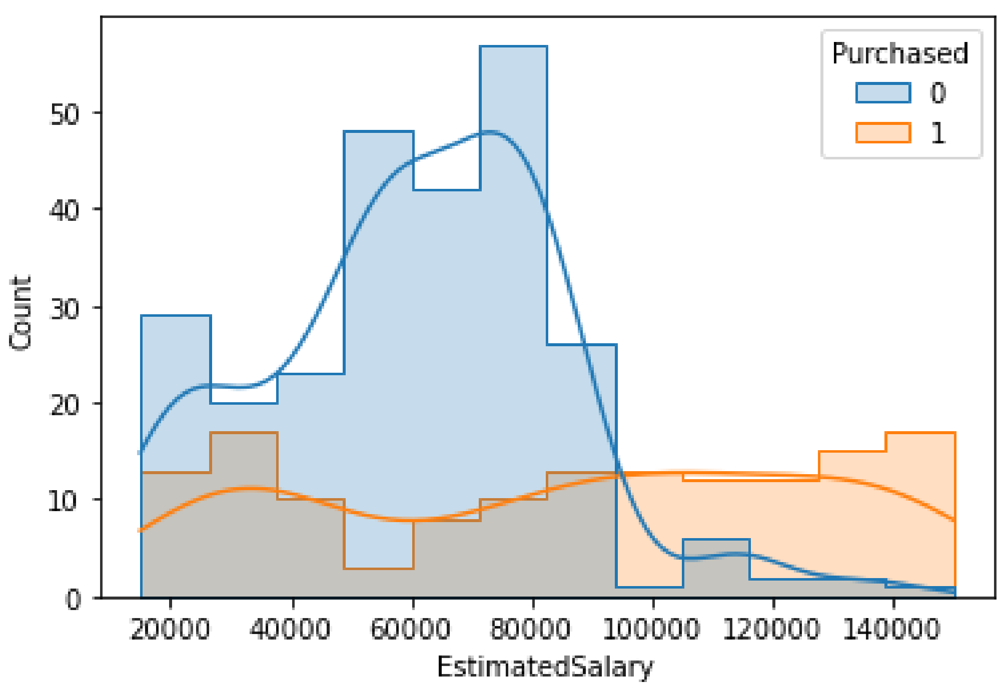
[246]:
```
ax = sns.boxplot(x="Purchased", y="age", hue="Gender", data=df, palette="Set3")
```



[247]:
```
sns.histplot(data=df, x="age", hue="Purchased",element="step",kde=True)
plt.show()
```



[248]:
```
sns.histplot(data=df, x="EstimatedSalary",␣
 ↪hue="Purchased",element="step",kde=True)
plt.show()
```



Predictive Models using (1) Logistic Regression and (2) Random Forest

The goal is to predict the "Purchase" behavior using "Age", "EstimatedSalary", and "Gender":

[249]:
```
X.head()
```

[249]:

	User ID	Gender	age	EstimatedSalary
0	15624510	Male	19	19000.0
1	15810944	Male	35	20000.0
2	15668575	Female	26	43000.0
3	15603246	Female	27	57000.0
4	15804002	Male	37	76000.0

[250]:
```
y.head()
```

[250]:
```
0    0
1    0
2    0
3    0
4    0
Name: Purchased, dtype: int64
```

Convert String value 'Gender' into dummy variables

[251]:
```
import pandas as pd
import numpy as np
```

[252]:
```
X = pd.get_dummies(X, columns=["Gender"])
```

[253]:
```
X.head()
```

[253]:

	User ID	age	EstimatedSalary	Gender_Female	Gender_Male
0	15624510	19	19000.0	0	1
1	15810944	35	20000.0	0	1
2	15668575	26	43000.0	1	0
3	15603246	27	57000.0	1	0
4	15804002	37	76000.0	0	1

[254]:
```
X.drop('User ID',axis=1,inplace=True)
X.head()
```

[254]:

	age	EstimatedSalary	Gender_Female	Gender_Male
0	19	19000.0	0	1
1	35	20000.0	0	1
2	26	43000.0	1	0

```
3   27          57000.0             1            0
4   37          76000.0             0            1
```

[255]:
```python
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   age              400 non-null    int64
 1   EstimatedSalary  400 non-null    float64
 2   Gender_Female    400 non-null    uint8
 3   Gender_Male      400 non-null    uint8
dtypes: float64(1), int64(1), uint8(2)
memory usage: 7.2 KB
```

[256]:
```python
X['Gender_Female'] = X['Gender_Female'].astype('int64')
```

[257]:
```python
X['Gender_Male'] = X['Gender_Male'].astype('int64')
```

[258]:
```python
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   age              400 non-null    int64
 1   EstimatedSalary  400 non-null    float64
 2   Gender_Female    400 non-null    int64
 3   Gender_Male      400 non-null    int64
dtypes: float64(1), int64(3)
memory usage: 12.6 KB
```

[260]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

[261]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

[261]:
```
LogisticRegression()
```

[262]:
```
y_pred = logreg.predict(X_test)
print(f'Accuracy of logistic regression classifier on test set:␣
↪{accuracy_score(y_test, y_pred)*100:.2f}%')
```

```
Accuracy of logistic regression classifier on test set: 70.00%
```

Random Forest Classification

[263]:
```
clf=RandomForestClassifier(n_estimators=100)
clf.fit(X_train,y_train)
```

[263]:
```
RandomForestClassifier()
```

[265]:
```
y_pred=clf.predict(X_test)
print(f'Accuracy of Random Forest classifier on test set:␣
↪{accuracy_score(y_test, y_pred)*100:.2f}%')
```

```
Accuracy of Random Forest classifier on test set: 91.25%
```

Feature Importance:

[267]:
```
print(clf.feature_importances_)
print(X.columns)
```

```
[0.48984002 0.49518898 0.00739093 0.00758006]
Index(['age', 'EstimatedSalary', 'Gender_Female', 'Gender_Male'],
dtype='object')
```

[269]:
```
# The feature importance sum up to 1
import pandas as pd
feature_imp = pd.Series(clf.feature_importances_, index=X.columns).
↪sort_values(ascending=False)
feature_imp
```

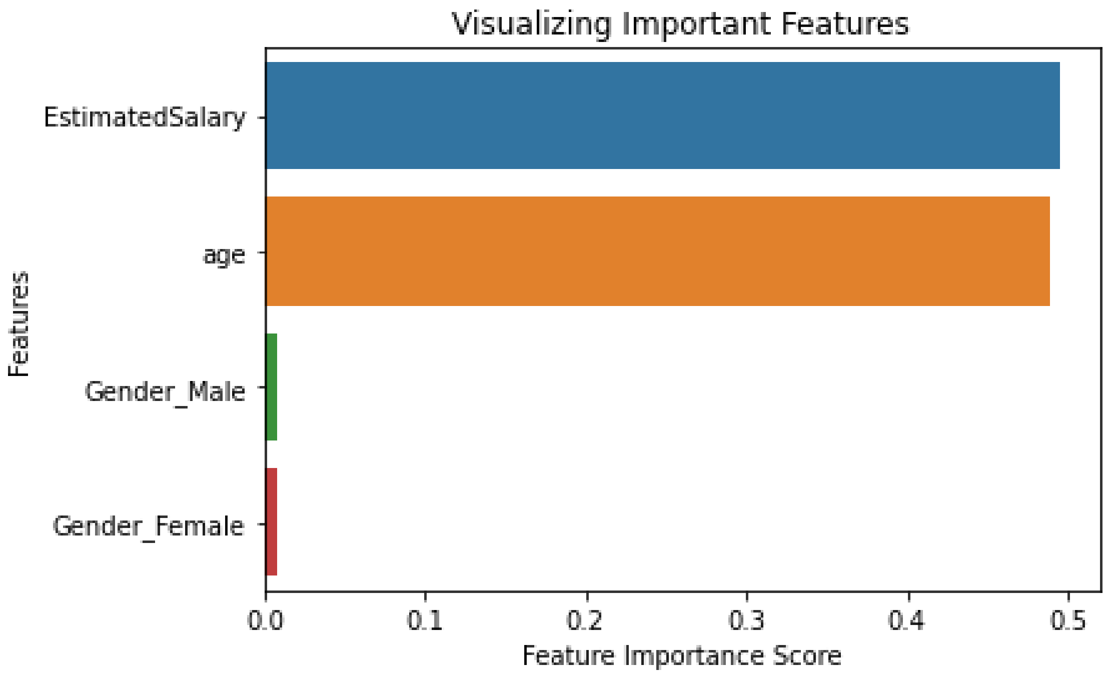
[269]:
```
EstimatedSalary    0.495189
age                0.489840
Gender_Male        0.007580
Gender_Female      0.007391
dtype: float64
```

Feature Importance in Visualization format

[271]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Creating a bar plot
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
```

```python
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.show()
```



Hyperparameter Tuning Random Forest: (1) number of decision trees and (2) number of predictors for splitting trees

The default values for (1) # of trees is "100" and (2) number of predictors is Sqare Root

[293]:
```python
# Grid searching key hyperparameters for RandomForestClassifier
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# define models and parameters
model = RandomForestClassifier()
n_estimators = [10, 100, 150, 200, 250, 300]
max_features = ['sqrt', 'log2', 'None']

# define grid search
grid = dict(n_estimators=n_estimators,max_features=max_features)
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
```

```
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv,␣
↪scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X, y)

# summarize results
print(f"Best Accuracy: {grid_result.best_score_*100: 0.2f}% with parameters:␣
↪{grid_result.best_params_}")
```

```
Best Accuracy:  88.17% with parameters: {'max_features': 'sqrt', 'n_estimators':
200}
```

[294]:
```
# Print out the performance, in Accurary, of other combinations:
means  = list(grid_result.cv_results_['mean_test_score'])
stds   = list(grid_result.cv_results_['std_test_score'])
params = list(grid_result.cv_results_['params'])

# Zip function will zip iterables, in this case 3 lists, into tuple
# and then perform "tuple unpacking" into components:
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
0.876667 (0.036477) with: {'max_features': 'sqrt', 'n_estimators': 10}
0.879167 (0.037823) with: {'max_features': 'sqrt', 'n_estimators': 100}
0.879167 (0.034055) with: {'max_features': 'sqrt', 'n_estimators': 150}
0.881667 (0.038424) with: {'max_features': 'sqrt', 'n_estimators': 200}
0.878333 (0.038586) with: {'max_features': 'sqrt', 'n_estimators': 250}
0.879167 (0.039441) with: {'max_features': 'sqrt', 'n_estimators': 300}
0.873333 (0.033187) with: {'max_features': 'log2', 'n_estimators': 10}
0.877500 (0.036856) with: {'max_features': 'log2', 'n_estimators': 100}
0.877500 (0.038243) with: {'max_features': 'log2', 'n_estimators': 150}
0.875833 (0.039651) with: {'max_features': 'log2', 'n_estimators': 200}
0.880000 (0.038406) with: {'max_features': 'log2', 'n_estimators': 250}
0.877500 (0.039843) with: {'max_features': 'log2', 'n_estimators': 300}
0.000000 (0.000000) with: {'max_features': 'None', 'n_estimators': 10}
0.000000 (0.000000) with: {'max_features': 'None', 'n_estimators': 100}
0.000000 (0.000000) with: {'max_features': 'None', 'n_estimators': 150}
0.000000 (0.000000) with: {'max_features': 'None', 'n_estimators': 200}
0.000000 (0.000000) with: {'max_features': 'None', 'n_estimators': 250}
0.000000 (0.000000) with: {'max_features': 'None', 'n_estimators': 300}
```

0.1 Let's try out the XGBoost model that won many Kaggle competition!

[296]:
```
!pip install xgboost
```

```
Defaulting to user installation because normal site-packages is not writeable
Looking in indexes:
http://nexus.opr.statefarm.org/repository/pypi.python.org/simple,
http://nexus.opr.statefarm.org/repository/python-internal/simple
```

```
Collecting xgboost
  Downloading http://nexus.opr.statefarm.org/repository/pypi.python.org/packages
/xgboost/1.3.3/xgboost-1.3.3-py3-none-manylinux2010_x86_64.whl (157.5 MB)
     |                 | 157.5 MB 5.8 MB/s eta 0:00:0101
Requirement already satisfied: numpy in /opt/conda/lib/python3.8/site-
packages (from xgboost) (1.19.4)
Requirement already satisfied: scipy in /opt/conda/lib/python3.8/site-packages
(from xgboost) (1.5.3)
Installing collected packages: xgboost
Successfully installed xgboost-1.3.3
```

[297]:
```
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

[300]:
```
# split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=2)
```

[301]:
```
print(X_train.shape)
print(X_test.shape)
```

```
(320, 4)
(80, 4)
```

[302]:
```
# fit model no training data
model = XGBClassifier()
model.fit(X_train, y_train)
```

```
/users/yycs/.local/lib/python3.8/site-packages/xgboost/sklearn.py:888:
UserWarning: The use of label encoder in XGBClassifier is deprecated and will be
removed in a future release. To remove this warning, do the following: 1) Pass
option use_label_encoder=False when constructing XGBClassifier object; and 2)
Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, …,
[num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

```
[17:40:53] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was changed
from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore
the old behavior.
```

[302]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=88, num_parallel_tree=1, random_state=0,
```

```
                   reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
                   tree_method='exact', validate_parameters=1, verbosity=None)
```

[305]:
```python
# make predictions for test data
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 86.25%
```

0.2 The performance of XGBoost (with default setting) is not better than Random Forest (Hyperparameter tuned).

0.3 The next thing to try is to tune its long list of hyperparameters.

[ ]:
```python

```