

```
import pandas as pd
import numpy as np
from scipy import stats
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

```
from google.colab import drive
drive.mount('/content/drive')
```

 Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

✓ Example 1

```
ex1=pd.read_csv('/content/drive/MyDrive/Colab Notebooks/BusStatUP/Datafiles/ARex1.csv')
ex1.columns = ex1.columns.str.replace(' ', '')
ex1.head()
```

	FamilyNo	SECClass	CarPrice	Incometax	Region
0	1	A1	1452851.190	1255879.520	West
1	2	A1	1366952.184	1130358.778	South
2	3	A1	1508334.296	1103459.366	South
3	4	A2	1496277.865	1059233.501	East
4	5	A2	1489735.296	1014217.110	South

Next steps: [Generate code with ex1](#) [View recommended plots](#)

```
X_parameters = ex1.columns
categorical_variables = ['SECClass', 'Region']
ex1new= pd.get_dummies( ex1[X_parameters], columns = categorical_variables, drop_first = True )
ex1new.head()
```

	FamilyNo	CarPrice	Incometax	SECClass_A2	SECClass_B1	SECClass_B2	SECClass_C	Region_North	Region_South	Region_West
0	1	1452851.190	1255879.520	False	False	False	False	False	False	True
1	2	1366952.184	1130358.778	False	False	False	False	False	True	False
2	3	1508334.296	1103459.366	False	False	False	False	False	True	False
3	4	1496277.865	1059233.501	True	False	False	False	False	False	False
4	5	1489735.296	1014217.110	True	False	False	False	False	True	False

Next steps: [Generate code with ex1new](#) [View recommended plots](#)

```
X_cols = ['Incometax', 'SECClass_A2', 'SECClass_B1', 'SECClass_B2', 'SECClass_C', 'Region_North', 'Region_South', 'Region_West']
X1 = ex1new[X_cols]
Y1=ex1new['CarPrice']
```

```
# Split the dataset into training and test set.
X_train, X_test, y_train, y_test = train_test_split(X1, Y1, test_size = 0.3, random_state = 42)
# create linear regression object
e6sk = LinearRegression()
# fit linear regression
e6sk.fit(X_train,y_train)
```

 **LinearRegression**
LinearRegression()

```
# get the slope and intercept of the line best fit.
print("Intercept          \t", e6sk.intercept_)
print("Coefficient        \t", e6sk.coef_)
print("rsquared           \t", e6sk.score(X1, Y1))

↗ Intercept          395218.4614350484
Coefficient        [ 9.23552850e-01 -1.03067106e+05 -1.51609775e+05 -1.04189445e+05
-2.09416338e+05 -1.03407601e+02 -1.31213237e+03 -5.41968822e+03]
rsquared           0.9070344360785898

print("rsquared      :", e6sk.score(X_test, y_test)) # r square in test data set
print('Train Score  : ', e6sk.score(X_train, y_train))
print('Test Score   : ', e6sk.score(X_test, y_test))

↗ rsquared      : 0.9144604558157053
Train Score    : 0.9038471539343143
Test Score     : 0.9144604558157053

# Predicting test and train data set e6sk model
predtest_y=e6sk.predict(X_test)
predtrain_y=e6sk.predict(X_train)

print("Train Data Set Mean squared error      : %.2f" % mean_squared_error(predtrain_y, y_train))
print("train Data Set Coefficient of determination : %.2f" % r2_score(predtrain_y, y_train))
print("Train Data Set Root Mean Square Error    : %.2f" % np.sqrt(metrics.mean_squared_error(predtrain_y, y_train)))

print("\n")
print("Test Data Set Mean squared error      : %.2f" % mean_squared_error(predtest_y, y_test))
print("test Data Set Coefficient of determination: %.2f" % r2_score(predtest_y, y_test))
print("Test Data Set Root Mean Square Error    : %.2f" % np.sqrt(metrics.mean_squared_error(predtest_y, y_test)) )

↗ Train Data Set Mean squared error      : 6762852060.50
Train Data Set Coefficient of determination : 0.89
Train Data Set Root Mean Square Error    : 82236.56

Test Data Set Mean squared error      : 6022188940.17
test Data Set Coefficient of determination: 0.91
Test Data Set Root Mean Square Error    : 77602.76
```

▼ Example 2

```
ex2=pd.read_csv('/content/drive/MyDrive/Colab Notebooks/BusStatUP/Datafiles/ARex2.csv')
ex2.columns = ex2.columns.str.replace(' ', '')
ex2.head()
```

↗

	FamilyNo	SECClass	CarPrice	Incometax	Region	OwnHouse	
0	1	A1	1480990	1184792.00	West	1	
1	2	A1	1437384	1149907.20	South	1	
2	3	A1	1484581	1128281.56	South	1	
3	4	A2	1426385	1126844.15	East	1	
4	5	A2	1477912	1108434.00	South	1	

il

Next steps: [Generate code with ex2](#) [View recommended plots](#)

```
variables = ex2.columns
variables

↗ Index(['FamilyNo', 'SECClass', 'CarPrice', 'Incometax', 'Region', 'OwnHouse'], dtype='object')

categorical_variables = ['SECClass', 'Region']
ex2new = pd.get_dummies( ex2[variables], columns = categorical_variables, drop_first = True )
ex2new.columns

↗ Index(['FamilyNo', 'CarPrice', 'Incometax', 'OwnHouse', 'SECClass_A2',
'SECClass_B1', 'SECClass_B2', 'SECClass_C', 'Region_North',
```

```
'Region_South', 'Region_West'],  
dtype='object')
```

```
X=ex2new[['Incometax', 'SECClass_B2']]  
Y=ex2new.OwnHouse  
import statsmodels.formula.api as sm
```

```
logi = sm.logit('OwnHouse~CarPrice+Incometax+SECClass_A2+ SECClass_B1+ SECClass_B2+SECClass_C+Region_North+Region_South+ Region_West', data=  
print(logi.summary())
```

```
➡ Optimization terminated successfully.  
Current function value: 0.508993  
Iterations 7
```

Logit Regression Results						
=====						
Dep. Variable:	OwnHouse	No. Observations:	1000			
Model:	Logit	Df Residuals:	990			
Method:	MLE	Df Model:	9			
Date:	Fri, 12 Jul 2024	Pseudo R-squ.:	0.2655			
Time:	02:54:35	Log-Likelihood:	-508.99			
converged:	True	LL-Null:	-692.99			
Covariance Type:	nonrobust	LLR p-value:	9.176e-74			
=====						
	coef	std err	z	P> z	[0.025	0.975]

Intercept	-3.9880	1.177	-3.387	0.001	-6.296	-1.680
SECClass_A2[T.True]	0.5174	0.390	1.327	0.185	-0.247	1.282
SECClass_B1[T.True]	-0.6531	0.532	-1.229	0.219	-1.695	0.389
SECClass_B2[T.True]	1.0089	0.394	2.558	0.011	0.236	1.782
SECClass_C[T.True]	0.4602	0.702	0.656	0.512	-0.915	1.835
Region_North[T.True]	-0.3506	0.221	-1.585	0.113	-0.784	0.083
Region_South[T.True]	-0.3177	0.221	-1.439	0.150	-0.750	0.115
Region_West[T.True]	-0.0492	0.218	-0.225	0.822	-0.477	0.378
CarPrice	-2.128e-07	1.16e-06	-0.184	0.854	-2.48e-06	2.06e-06
Incometax	7.461e-06	2.13e-06	3.507	0.000	3.29e-06	1.16e-05
=====						

```
logi = sm.logit('OwnHouse~Incometax+SECClass_B2',data=ex2new).fit()  
print(logi.summary())
```

```
➡ Optimization terminated successfully.  
Current function value: 0.529215  
Iterations 6
```

Logit Regression Results						
=====						
Dep. Variable:	OwnHouse	No. Observations:	1000			
Model:	Logit	Df Residuals:	997			
Method:	MLE	Df Model:	2			
Date:	Fri, 12 Jul 2024	Pseudo R-squ.:	0.2363			
Time:	02:54:35	Log-Likelihood:	-529.21			
converged:	True	LL-Null:	-692.99			
Covariance Type:	nonrobust	LLR p-value:	7.505e-72			
=====						
	coef	std err	z	P> z	[0.025	0.975]

Intercept	-4.0680	0.293	-13.875	0.000	-4.643	-3.493
SECClass_B2[T.True]	0.8833	0.303	2.913	0.004	0.289	1.478
Incometax	7.166e-06	5.21e-07	13.756	0.000	6.14e-06	8.19e-06
=====						

```
X=ex2new[['Incometax', 'SECClass_B2']]  
Y=ex2new.OwnHouse
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state = 42)
```

```
X_train.head()
```

	Incometax	SECClass_B2	
541	525293.60	False	
440	575285.26	False	
482	551755.40	False	
422	581646.98	False	
778	418065.00	False	

Next steps:

[Generate code with X_train](#)
[View recommended plots](#)

y_train.head()

541	0
440	0
482	1
422	1
778	0
Name: OwnHouse, dtype: int64	

```
# instantiate the model (using the default parameters)
modelf = LogisticRegression(random_state=16)
```

```
# fit the model with data
modelf.fit(X_train, y_train)
```

```
y_pred = modelf.predict(X_test)
```

```
y_pred_traindf = pd.DataFrame( { "actual": y_train, "predicted_prob": modelf.predict(X_train) } )
y_pred_traindf.head(125)
```

	actual	predicted_prob	
541	0	1	
440	0	1	
482	1	1	
422	1	1	
778	0	1	
...	...	...	
18	1	1	
715	1	1	
483	1	1	
568	0	1	
433	0	1	
125 rows × 2 columns			

Next steps:

[Generate code with y_pred_traindf](#)
[View recommended plots](#)

```
testdf = pd.DataFrame( { "actual": y_test, "predicted_prob": modelf.predict(X_test) } )
testdf.head(5)
```

	actual	predicted_prob	
521	1	1	
737	0	1	
740	0	1	
660	0	1	
411	1	1	

Next steps:

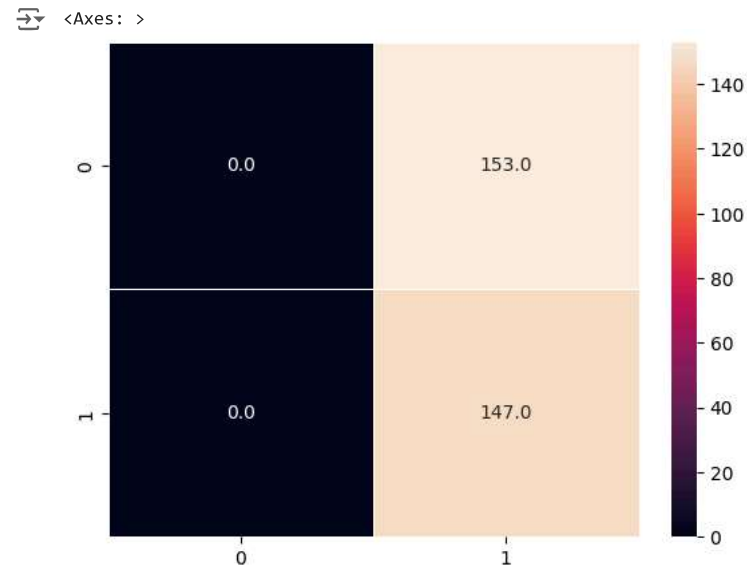
[Generate code with testdf](#)
[View recommended plots](#)

```
# import the metrics class
from sklearn import metrics
```

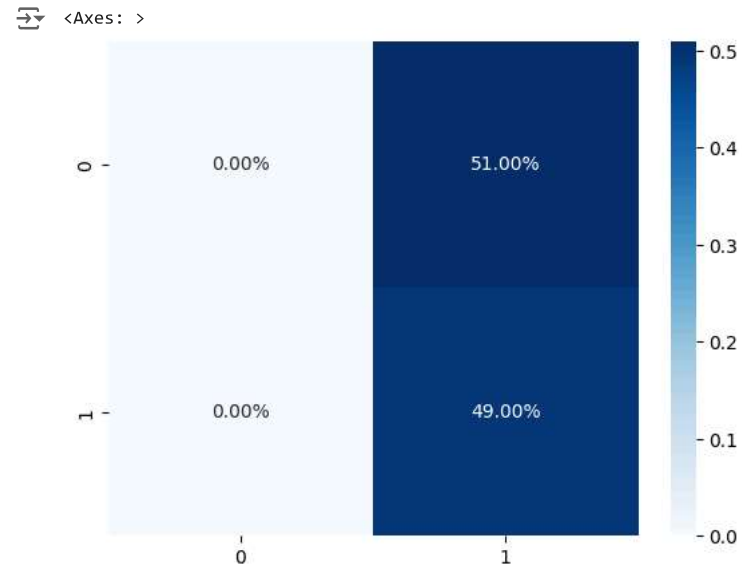
```
cm = metrics.confusion_matrix(y_test, y_pred)
cm
```

```
array([[ 0, 153],
       [ 0, 147]])
```

```
sns.heatmap(cm, annot=True,linewidths=.5, fmt= '.1f')
```



```
sns.heatmap(cm/np.sum(cm), annot=True, fmt='.2%', cmap='Blues')
```



```
cf=metrics.confusion_matrix(testdf.actual, testdf.predicted_prob).ravel()
tn, fp, fn, tp = cf
print('True Positive', tp)
print('True Negative', tn)
print('False Positive', fp)
print('False Negative', fn)
```

```
True Positive 147
True Negative 0
False Positive 153
False Negative 0
```

```
ps=metrics.precision_score(testdf.actual, testdf.predicted_prob)
r= tp/(tp+fn)
as1=metrics.accuracy_score(testdf.actual, testdf.predicted_prob)

print('accuracy_score', as1)
print('precision_score', ps)

print('recall', r)
```

```
↗ accuracy_score 0.49
precision_score 0.49
recall 1.0
```

```
X=ex2new[['FamilyNo', 'CarPrice', 'Incometax', 'SECClass_A2', 'SECClass_B1', 'SECClass_B2', 'SECClass_C', 'Region_North', 'Region_South', 'Region_West']]
Y=ex2new.OwnHouse
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state = 42)
```

```
# instantiate the model (using the default parameters)
modelf = LogisticRegression(random_state=16)
```

```
# fit the model with data
modelf.fit(X_train, y_train)
```

```
y_pred = modelf.predict(X_test)
```

```
X_train.head()
```

```
↗
```

	FamilyNo	CarPrice	Incometax	SECClass_A2	SECClass_B1	SECClass_B2	SECClass_C	Region_North	Region_South	Region_West	
541	542	808144	525293.60	True	False	False	False	False	False	False	
440	441	788062	575285.26	True	False	False	False	False	False	True	
482	483	811405	551755.40	True	False	False	False	False	False	True	
422	423	736262	581646.98	True	False	False	False	True	False	False	
778	779	696775	418065.00	False	True	False	False	False	False	False	

Next steps: [Generate code with X_train](#) [View recommended plots](#)

```
testdf = pd.DataFrame( { "actual": y_test, "predicted_prob": modelf.predict(X_test) } )
testdf.head(5)
```

```
↗
```

	actual	predicted_prob	
521	1	0	
737	0	0	
740	0	0	
660	0	0	
411	1	1	

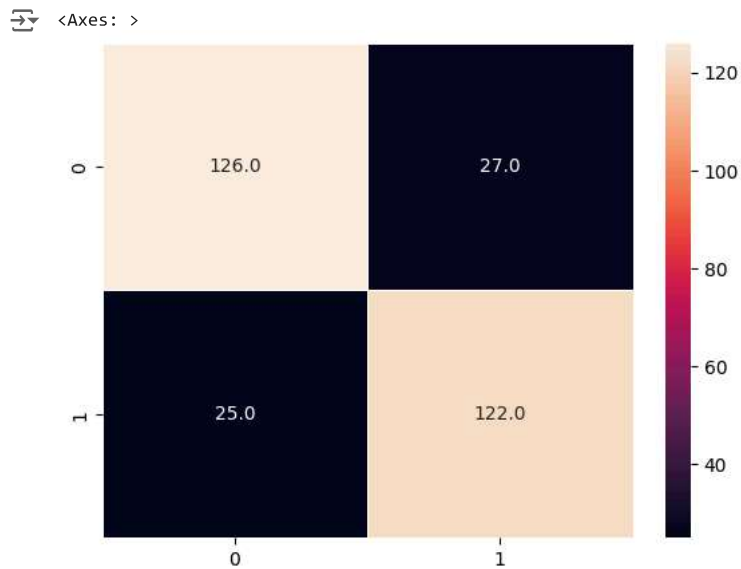
Next steps: [Generate code with testdf](#) [View recommended plots](#)

```
# import the metrics class
from sklearn import metrics
```

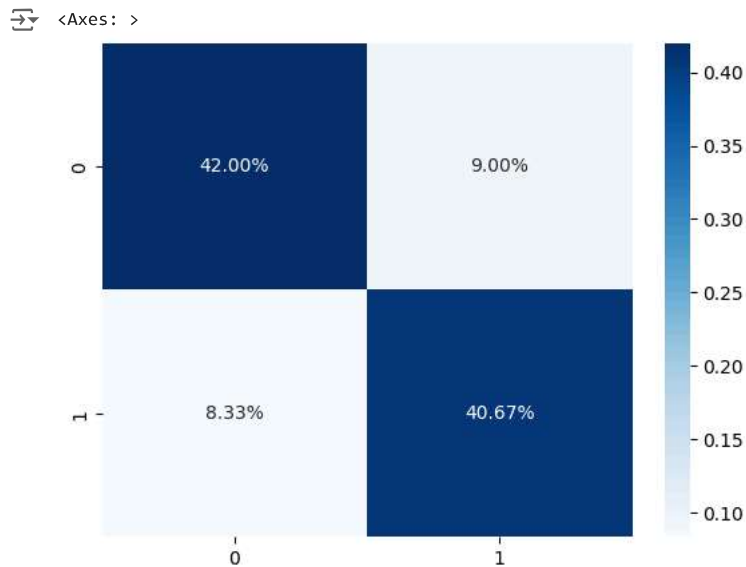
```
cm = metrics.confusion_matrix(y_test, y_pred)
cm
```

```
↗ array([[126, 27],
        [ 25, 122]])
```

```
sns.heatmap(cm, annot=True,linewidths=.5, fmt= '.1f')
```



```
sns.heatmap(cm/np.sum(cm), annot=True, fmt='.2%', cmap='Blues')
```



```
cf=metrics.confusion_matrix(testdf.actual, testdf.predicted_prob).ravel()
tn, fp, fn, tp = cf
print('True Positive', tp)
print('True Negative', tn)
print('False Positive', fp)
print('False Negative', fn)
```

True Positive 122
True Negative 126
False Positive 27
False Negative 25

```
ps=metrics.precision_score(testdf.actual, testdf.predicted_prob)
r= tp/(tp+fn)
as1=metrics.accuracy_score(testdf.actual, testdf.predicted_prob)
```

```
print('accuracy_score', as1)
print('precision_score', ps)
```

```
print('recall', r)
```

accuracy_score 0.8266666666666667
precision_score 0.8187919463087249
recall 0.8299319727891157

Start coding or [generate](#) with AI.