Feature Engineering-ENCODING

LabelEncoder & OneHotEncoder both

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
```

LabelEncoder

In [ ]:
```
labelencoder = LabelEncoder()

label_data = labelencoder.fit_transform(Data['Types'])
label_data
```

In [ ]:
```
Data['label_types'] = labelencoder.inverse_transform(Data['label'])
```

In [ ]:
```
df['education'] = labelencoder.fit_transform(df['education'])
```

OneHotEncoder

In [ ]:
```
dummies = pd.get_dummies(data.Department)
```

In [ ]:
```
dum_df= pd.get_dummies(df, columns=["relationship"],prefix=["Rtype_"])
```

Feature Engineering-SCALING

Normalization (MinMaxScaler)

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

matplotlib.style.use('fivethirtyeight')
%matplotlib inline
```

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [ ]:
```
scaling = MinMaxScaler()
```

In [ ]:
```
df[['Pregnancies_','Glucose_']]= scaling.fit_transform(df[['Pregnancies','Glucose']]
```

Standardization (StandardScaler)

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

In [ ]:
```
std_scal = StandardScaler()
```

In [ ]:
```
df[['BloodPressure_','SkinThickness_']]= std_scal.fit_transform(df[['BloodPressure',
```

RobustScaler

In [ ]:
```
from sklearn.preprocessing import RobustScaler
```

In [ ]:
```
Ro_scaler = RobustScaler()
```

In [ ]:
```
df[['Insulin_','BMI_']]= Ro_scaler.fit_transform(df[['Insulin','BMI']])
```

Simple Linear Regression

In [ ]:
```
import pandas as pd # for data frame
import numpy as np # for mathemetical operation
import matplotlib.pyplot as plt # for vizualization
%matplotlib inline
from sklearn.linear_model import LinearRegression #  For regression in Machine Learn
from sklearn.metrics import r2_score # For Accuracy
```

In [ ]:
```
regression_model = LinearRegression()
```

In [ ]:
```
regression_model.fit(area,price)
```

In [ ]:
```
price_predicted = regression_model.predict([[3000]])
price_predicted
```

In [ ]:
```
r2=r2_score(price,price_predicted)
r2
```

In [ ]:
```
print('Slope:' ,regression_model.coef_)
print('Intercept:', regression_model.intercept_)
print('R2 score: ', r2)
```

Multiple Linear Regression

In [ ]:
```
import numpy as np # for data frame
import pandas as pd # for mathemetical operation
import matplotlib.pyplot as plt # for vizualization
%matplotlib inline
import seaborn as sns
```

```
from sklearn.linear_model import LinearRegression #  For regression in Machine Learn
from sklearn.metrics import r2_score # For Accuracy
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=10)
```

In [ ]:
```
from sklearn.linear_model import LinearRegression
```

In [ ]:
```
regression_model=LinearRegression()
```

In [ ]:
```
regression_model.fit(X_train,Y_train)
```

In [ ]:
```
Predicted_Price=regression_model.predict(X_test)
Predicted_Price
```

In [ ]:
```
regression_model.score(X_test,Y_test)
```

In [ ]:
```
from sklearn.metrics import mean_squared_error,r2_score
```

In [ ]:
```
mse=mean_squared_error(Y_test,Predicted_Price)
r2 = r2_score(Y_test,Predicted_Price)
```

In [ ]:
```
print('Slope:' ,regression_model.coef_)
print('Intercept:', regression_model.intercept_)
print('Root mean squared error: ', mse)
print('R2 score: ', r2)
```

Ridge Overfitting Prevention Technique

In [ ]:
```
ridge_reg= Ridge(alpha=50, max_iter=100,)
ridge_reg.fit(X_train, Y_train)
```

In [ ]:
```
ridge_reg.score(X_test, Y_test)
```

In [ ]:
```
ridge_reg.score(X_train,Y_train)
```

Lasso Overfitting Prevention Technique

In [ ]:
```
lasso_reg = Lasso(alpha=50, max_iter=100)
lasso_reg.fit(X_train,Y_train)
```

In [ ]:
```
lasso_reg.score(X_test,Y_test)
```

In [ ]:
```
lasso_reg.score(X_train,Y_train)
```

Logistic Regression

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [ ]:
```
#feature selection
df.corr()
sns.heatmap(df.corr(), xticklabels=df.corr().columns, yticklabels=df.corr().columns,
```

In [ ]:
```
df = pd.get_dummies(df,columns=['salary'], prefix="salary")
```

In [ ]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression
```

In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(x,y,train_size=0.3,random_state=
```

In [ ]:
```
model = LogisticRegression(random_state=10)
model.fit(X_train, y_train)
```

In [ ]:
```
model.score(X_test,y_test)
```

In [ ]:
```
model.score(X_train,y_train)
```

In [ ]:
```
y_pred = model.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import confusion_matrix
```

In [ ]:
```
con_mat = confusion_matrix(y_test,y_pred)
con_mat
```

In [ ]:
```
plt.figure(figsize = (10,7))
sns.heatmap(con_mat, annot=True,fmt='g')
plt.xlabel('Predicted label')
plt.ylabel('Actual label')
```

In [ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,f1_score
```

In [ ]:
```
print("Accuracy:",accuracy_score(y_test, y_pred))
print("Precision:",precision_score(y_test, y_pred))
print("Recall:",recall_score(y_test, y_pred))
print("f1_score:",f1_score(y_test, y_pred))
```

In [ ]:
```
from sklearn.metrics import roc_curve,roc_auc_score
```

In [ ]:
```
y_pred_proba = model.predict_proba(X_test)[::,1]
fpr, tpr, threshold = roc_curve(y_test,  y_pred_proba)
auc = roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.xlabel("False Positive Rate")
plt.ylabel(" Positive Rate")

plt.show()
```

In [ ]:
```
from sklearn.metrics import classification_report
```

In [ ]:
```
print(classification_report(y_test,y_pred))
```

Outlier Removal

Discussion Related With Outliers And Impact On Machine Learning!! Which Machine LEarning Models Are Sensitive To Outliers?

Naivye Bayes Classifier--- Not Sensitive To Outliers

SVM-------- Not Sensitive To Outliers

Linear Regression---------- Sensitive To Outliers

Logistic Regression------- Sensitive To Outliers

Decision Tree Regressor or Classifier---- Not Sensitive

Ensemble(RF,XGboost,GB)------- Not Sensitive

KNN-------------------------- Not Sensitive

Kmeans----------------------- Sensitive

Hierarichal------------------- Sensitive

PCA------------------------- Sensitive

Neural Networks-------------- Sensitive

Using "Z score"

In [ ]:
```
# Using Z score

Formula for Z score = (Observation — Mean)/Standard Deviation
```

```
Data point that falls outside of 3 standard deviations. we can use a z score and if


z = (X − μ) / σ
```

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
df['zscore'] = ( df.Score - df.Score.mean() ) / df.Score.std()
df.head(20)
```

In [ ]:
```
df[df['zscore']>3]
```

OR

In [ ]:
```
#one method
def detect_outliers(data):

    threshold=3
    mean = np.mean(data)
    std =np.std(data)


    for i in data:
        z_score= (i - mean)/std
        if np.abs(z_score) > threshold:
            outliers.append(i)
    return outliers
```

Outlier detection and removal using 3 standard deviation

In [ ]:
```
upper_limit = df.Score.mean() + 3*df.Score.std()
upper_limit
```

In [ ]:
```
lower_limit = df.Score.mean() -3*df.Score.std()
lower_limit
```

In [ ]:
```
df[(df.Score>upper_limit) | (df.Score<lower_limit)]
```

In [ ]:
```
sorted(df['Score'])
```

In [ ]:
```
quantile1, quantile3= np.percentile(df['Score'],[25,75])
```

In [ ]:
```
## Find the IQR
```

```
iqr=quantile3-quantile1
print(iqr)
```

In [ ]:
```
## Find the lower bound value and the higher bound value

lower_bound_val = quantile1 -(1.5 * iqr)
upper_bound_val = quantile3 +(1.5 * iqr)
```

In [ ]:
```
df[(df.Score<lower_bound_val)|(df.Score>upper_bound_val)]
```

Decsion Tree

In [3]:
```
import numpy as np # linear algebra
import pandas as pd # data processing
import matplotlib.pyplot as plt # data visualization
%matplotlib inline
import seaborn as sns # statistical data visualization


from sklearn.tree import DecisionTreeClassifier #Decision Tree
from sklearn.model_selection import train_test_split#Train Test Split
from sklearn import metrics
from sklearn import tree
```

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
```

In [ ]:
```
input= input.apply(LabelEncoder().fit_transform)
```

In [ ]:
```
X_train, X_test, Y_train, Y_test = train_test_split(input,target, test_size = 0.3, r
```

In [ ]:
```
from sklearn import tree
model = tree.DecisionTreeClassifier()
```

In [ ]:
```
model.fit(X_train,Y_train)
```

In [ ]:
```
model.score(X_train,Y_train)
```

In [ ]:
```
model.score(X_test,Y_test)
```

In [ ]:
```
y_pred  = model.predict(X_test)
y_pred
```

In [ ]:
```
from sklearn import metrics
```

In [ ]:
```
metrics.confusion_matrix(Y_test, y_pred)
```

In [ ]:
```
from sklearn import tree
text_representation = tree.export_text(model)
print(text_representation)
```

In [ ]:
```
plt.figure(figsize=(50,50))

from sklearn import tree

tree.plot_tree(model,feature_names=input.columns,
                   class_names=['0','1'],
                   filled=True)
```

In [ ]:
```
# confusion matrix,accuracy,classification_report in sklearn
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

In [ ]:
```
matrix = confusion_matrix(Y_test,y_pred, labels=[1,0])
print('Confusion matrix : \n',matrix)
```

In [ ]:
```
accuracy = metrics.accuracy_score(Y_test,y_pred)
```

In [ ]:
```
report=classification_report(Y_test,y_pred)
```

In [ ]:
```
from sklearn import datasets

# Prepare the data data
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

In [ ]:
```
clf = DecisionTreeClassifier(random_state=1234)
model = clf.fit(X, y)
```

In [ ]:
```
ext_representation = tree.export_text(clf)
print(text_representation)
```

In [ ]:
```
fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(clf,
                   feature_names=iris.feature_names,
                   class_names=iris.target_names,
                   filled=True)
```

Support Vector Machine (SVM)

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

```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
sns.pairplot(df)
```

In [ ]:
```
X = df.drop(['target'], axis=1)

y = df['target']
```

In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_st
```

In [ ]:
```
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

In [ ]:
```
# import SVC classifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

In [ ]:
```
# instantiate classifier with default hyperparameters
svc=SVC(random_state=10)
svc.fit(X_train,y_train)
```

In [ ]:
```
# make predictions on test set
y_pred=svc.predict(X_test)
y_pred_train=svc.predict(X_train)


print('Model accuracy score  without scaling  {0:0.4f}'. format(accuracy_score(y_tes
print('Model accuracy score  without scaling  {0:0.4f}'. format(accuracy_score(y_tra
```

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

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

In [ ]:
```
# import SVC classifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

In [ ]:
```
# instantiate classifier with default hyperparameters
svc=SVC()
svc.fit(X_train,y_train)
```

In [ ]:

```
# make predictions on test set
y_pred=svc.predict(X_test)


print('Model accuracy score rbf  {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

In [ ]:

```
# instantiate classifier with rbf kernel and C=100
svc_1=SVC(C=100)


# fit classifier to training set
svc_1.fit(X_train,y_train)


# make predictions on test set
y_pred_rbf=svc_1.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with rbf and c= 100  : {0:0.4f}'. format(accuracy_score(
```

kernel : string, optional (default='rbf')

```
Specifies the kernel type to be used in the algorithm.
It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or
a callable.
If none is given, 'rbf' will be used. If a callable is given it is
used to pre-compute the kernel matrix from data matrices; that matrix
should be an array of shape ``(n_samples, n_samples)
```

Run SVM with polynomial kernel

In [ ]:

```
# instantiate classifier with polynomial kernel and C=1.0
poly_svc=SVC(kernel='poly', C=1.0)


# fit classifier to training set
poly_svc.fit(X_train,y_train)


# make predictions on test set
y_pred_poly = poly_svc.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with polynomial kernel : {0:0.4f}'. format(accuracy_scor
```

In [ ]:

```
print('Training set score: {:.4f}'.format(svc.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(svc.score(X_test, y_test)))
```

In [ ]:

```
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix
```

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

In [ ]:
```
#convert into dataframe
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0']
                                  index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

In [ ]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

Naive Bayes

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
df['scaled_Amount'] = StandardScaler().fit_transform(df['Amount'].values.reshape(-1,
df = df.drop(['Amount'],axis=1)
```

In [ ]:

In [ ]:
```
from sklearn.model_selection import train_test_split
```

In [ ]:
```
X_train , X_test , y_train , y_test = train_test_split(X,y,test_size = 0.3,random_st
```

In [ ]:
```
from sklearn.naive_bayes import GaussianNB
```

In [ ]:
```
model = GaussianNB()
model.fit(X_train,y_train)
```

In [ ]:
```
model.score(X_train,y_train)
```

In [ ]:
```
model.score(X_test,y_test)
```

In [ ]:
```
model.predict(X_test[0:5])
```

In [ ]:
```
model.predict_proba(X_test[:5])
```

In [ ]:
```
y_pred = model.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import accuracy_score, confusion_matrix,classification_report
```

In [ ]:
```
accuracy_score(y_test,y_pred)
```

In [ ]:
```
confusion_matrix(y_test,y_pred)
```

In [ ]:
```
print(classification_report(y_test,y_pred))
```

KNN

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('ggplot')
import seaborn as sns
sns.set()
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [ ]:
```
sns.pairplot(df, hue = 'Outcome')
```

In [ ]:
```
plt.figure(figsize=(12,10))  # on this line I just set the size of figure to 12 by 1
p=sns.heatmap(df.corr(), annot=True,cmap ='RdYlGn')  # seaborn has very simple solut
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
```

In [ ]:
```
#importing train_test_split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42,
```

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

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

In [ ]:
```
X_train = pd.DataFrame(X_train, columns=[cols])
X_test = pd.DataFrame(X_test, columns=[cols])
```

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [ ]:
```
model = KNeighborsClassifier(1)
model.fit(X_train,y_train)
```

In [ ]:
```
model.score(X_train,y_train)
```

In [ ]:
```
model.score(X_test,y_test)
```

OR

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier


test_scores = []
train_scores = []

for i in range(1,15):

    model = KNeighborsClassifier(i)
    model.fit(X_train,y_train)

    train_scores.append(model.score(X_train,y_train))
    test_scores.append(model.score(X_test,y_test))
```

In [ ]:
```
plt.figure(figsize=(12,5))
p = sns.lineplot(range(1,15),train_scores,marker='*',label='Train Score')
p = sns.lineplot(range(1,15),test_scores,marker='o',label='Test Score')
```

In [ ]:
```
#Setup a knn classifier with k neighbors
knn = KNeighborsClassifier(14)

knn.fit(X_train,y_train)
```

In [ ]:
```
knn.score(X_train,y_train)
```

In [ ]:
```
knn.score(X_test,y_test)
```

In [ ]:
```
y_pred = knn.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import accuracy_score , confusion_matrix,classification_report
```

In [ ]:
```
accuracy_score(y_test,y_pred)
```

In [ ]:
```
confusion_matrix(y_test,y_pred)
```

In [ ]:
```
print(classification_report(y_test,y_pred))
```

Random Forest

In [ ]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # data visualization
%matplotlib inline
import seaborn as sns # statistical data visualization
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_st
```

In [ ]:
```
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

In [ ]:
```
rfc = RandomForestClassifier(random_state=0)
rfc.fit(X_train, y_train)
```

In [ ]:
```
y_pred = rfc.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import accuracy_score
```

In [ ]:
```
y_pred_train = rfc.predict(X_train)
```

In [ ]:
```
accuracy_score(y_train,y_pred_train)
```

In [ ]:
```
accuracy_score(y_test,y_pred)
```

In [ ]:
```
rfc_100 = RandomForestClassifier(n_estimators=1000, random_state=0)


rfc_100.fit(X_train, y_train)
```

In [ ]:
```
# Predict on the test set results

y_pred_100 = rfc_100.predict(X_test)
y_pred_100_train = rfc_100.predict(X_train)



# Check accuracy score

print('Model accuracy score with 1000 decision-trees : {0:0.4f}'. format(accuracy_sc
print('Model accuracy score with 1000 decision-trees : {0:0.4f}'. format(accuracy_sc
```

In [ ]:
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_100)

print('Confusion matrix\n\n', cm)
```

In [ ]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_100))
```

In [ ]:
```
rfc.feature_importances_
```

In [ ]:
```
X_train.columns
```

Unsupervised Machine Learning

K-Means

In [ ]:
```
from sklearn.cluster import KMeans
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
```

In [ ]:
```
km = KMeans(n_clusters=3,random_state=10)
km.fit(df)
```

In [ ]:
```
y_predicted = km.fit_predict(df[['Age','Income($)']])
y_predicted
```

In [ ]:
```
km.cluster_centers_
```

In [ ]:
```
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
```

In [ ]:
```
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker=
plt.legend()
```

In [ ]:
```
from sklearn.metrics import silhouette_score
```

In [ ]:
```
score = silhouette_score(df, y_predicted)
score
```

Hierarchical Clustering with AgglomerativeClustering & Dendogram

In [ ]:
```
from scipy.cluster.hierarchy import linkage,dendrogram
merg = linkage(df, method = "ward")
dendrogram(merg, leaf_rotation = 0)
plt.xlabel("data points")
plt.ylabel("euclidean distance")
plt.show()
```

In [ ]:
```
from sklearn.cluster import AgglomerativeClustering
```

In [ ]:
```
hc = AgglomerativeClustering(n_clusters = 3, affinity = "euclidean", linkage = "ward
cluster = hc.fit_predict(df)
```

In [ ]:
```
df.label.value_counts()
```

In [ ]:
```
from sklearn.metrics import silhouette_score
```

In [ ]:
```
score_agg = silhouette_score(df, cluster)
score_agg
```

Extra Useful Infromation

In [ ]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
pd.set_option('display.max_columns', None)
```

In [ ]:
```

```