```
#1.Download the Employee Attrition Dataset
#https://www.kaggle.com/datasets/patelprashant/employee-attrition
#2.Perfrom Data Preprocessing
#3.Model Building using Logistic Regression and Decision Tree
#4.Calculate Performance metrics


# performing linear algebra
import numpy as np

# data processing
import pandas as pd

# visualisation
import matplotlib.pyplot as plt
import seaborn as sns


df = pd.read_csv("Employee-Attrition.csv")
print (dataset.head)
```

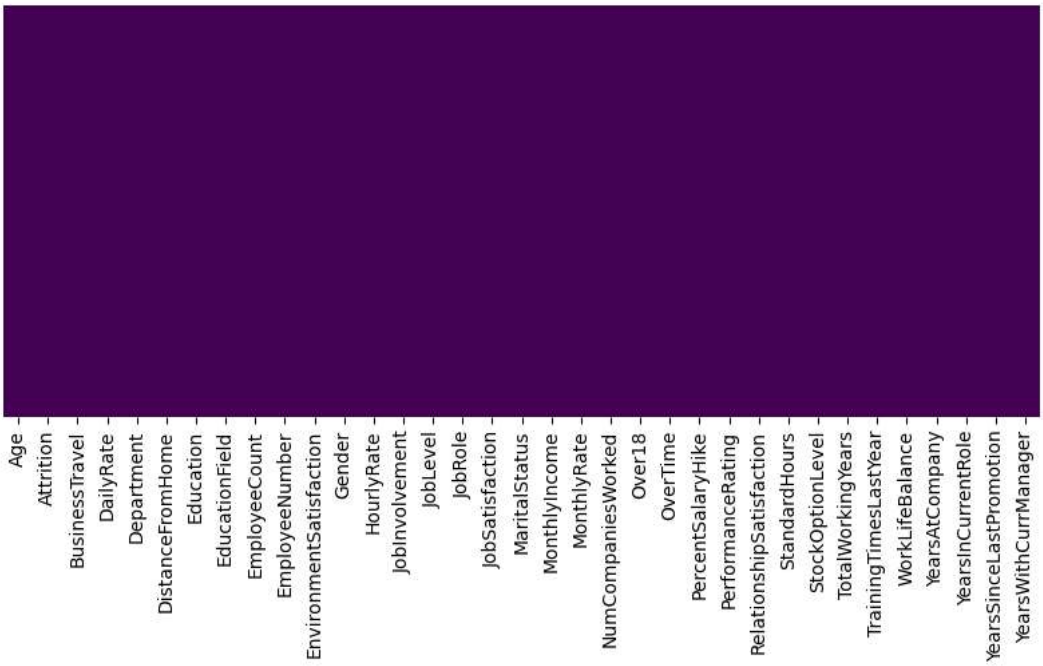
```
     1466                6          1        Medical             1
     1467                4          3  Life Sciences             1
     1468                2          3        Medical             1
     1469                8          3        Medical             1

           EmployeeNumber  ...  RelationshipSatisfaction StandardHours  \
     0                  1  ...                         1            80
     1                  2  ...                         4            80
     2                  4  ...                         2            80
     3                  5  ...                         3            80
     4                  7  ...                         4            80
     ...              ...  ...                       ...           ...
     1465            2061  ...                         3            80
     1466            2062  ...                         1            80
     1467            2064  ...                         2            80
     1468            2065  ...                         4            80
     1469            2068  ...                         1            80

           StockOptionLevel  TotalWorkingYears  TrainingTimesLastYear  \
     0                    0                  8                      0
     1                    1                 10                      3
     2                    0                  7                      3
     3                    0                  8                      3
     4                    1                  6                      3
     ...                ...                ...                    ...
     1465                 1                 17                      3
     1466                 1                  9                      5
     1467                 1                  6                      0
     1468                 0                 17                      3
     1469                 0                  6                      3

          WorkLifeBalance  YearsAtCompany YearsInCurrentRole  \
     0                  1               6                  4
     1                  3              10                  7
     2                  3               0                  0
     3                  3               8                  7
     4                  3               2                  2
     ...              ...             ...                ...
     1465               3               5                  2
     1466               3               7                  7
     1467               3               6                  2
     1468               2               9                  6
     1469               4               4                  3

           YearsSinceLastPromotion  YearsWithCurrManager
     0                           0                     5
     1                           1                     7
     2                           0                     0
     3                           3                     0
     4                           2                     2
     ...                       ...                   ...
     1465                        0                     3
     1466                        1                     7
     1467                        0                     3
     1468                        0                     8
     1469                        1                     2

     [1470 rows x 35 columns]>
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Age                       1470 non-null   int64 
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64 
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64 
 6   Education                 1470 non-null   int64 
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64 
 9   EmployeeNumber            1470 non-null   int64 
 10  EnvironmentSatisfaction   1470 non-null   int64 
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64 
 13  JobInvolvement            1470 non-null   int64 
 14  JobLevel                  1470 non-null   int64 
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64 
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64 
 19  MonthlyRate               1470 non-null   int64 
 20  NumCompaniesWorked        1470 non-null   int64 
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64 
 24  PerformanceRating         1470 non-null   int64 
 25  RelationshipSatisfaction  1470 non-null   int64 
 26  StandardHours             1470 non-null   int64 
 27  StockOptionLevel          1470 non-null   int64 
 28  TotalWorkingYears         1470 non-null   int64 
 29  TrainingTimesLastYear     1470 non-null   int64 
 30  WorkLifeBalance           1470 non-null   int64 
 31  YearsAtCompany            1470 non-null   int64 
 32  YearsInCurrentRole        1470 non-null   int64 
 33  YearsSinceLastPromotion   1470 non-null   int64 
 34  YearsWithCurrManager      1470 non-null   int64 
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

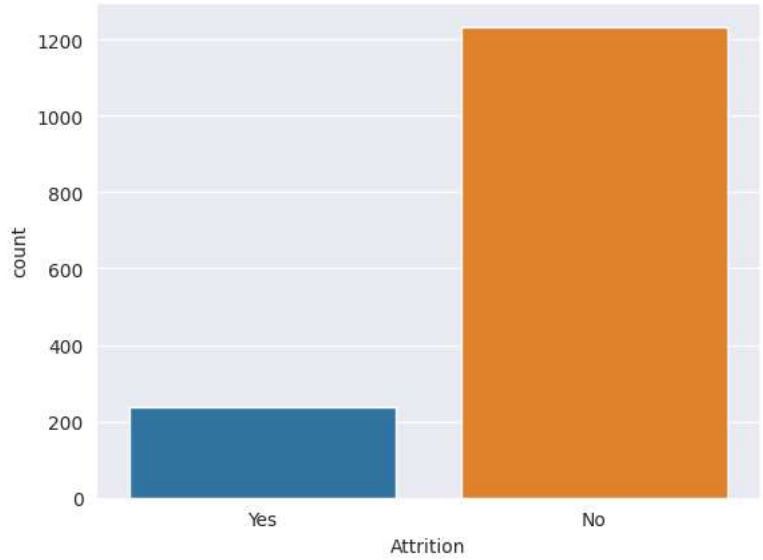
```
# heatmap to check the missing value
plt.figure(figsize =(10, 4))
sns.heatmap(dataset.isnull(), yticklabels = False, cbar = False, cmap ='viridis')
```

```
<Axes: >
```



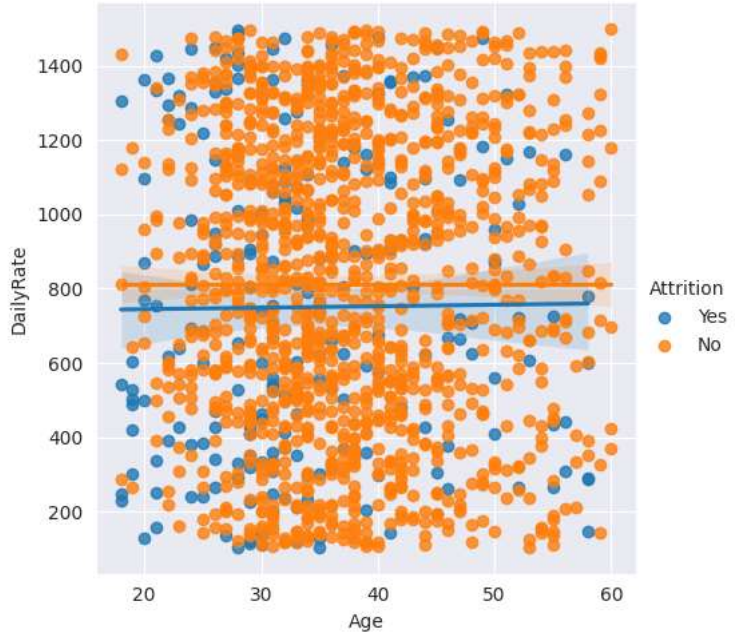
```
sns.set_style('darkgrid')
sns.countplot(x ='Attrition', data = dataset)
```

```
<Axes: xlabel='Attrition', ylabel='count'>
```




```
sns.lmplot(x = 'Age', y = 'DailyRate', hue = 'Attrition', data = dataset)
```

```
<seaborn.axisgrid.FacetGrid at 0x7edab4f6af20>
```




```
plt.figure(figsize =(10, 6))
sns.boxplot(y ='MonthlyIncome', x ='Attrition', data = dataset)
```

```
<Axes: xlabel='Attrition', ylabel='MonthlyIncome'>
```


```
dataset.drop('EmployeeCount', axis = 1, inplace = True)
dataset.drop('StandardHours', axis = 1, inplace = True)
dataset.drop('EmployeeNumber', axis = 1, inplace = True)
dataset.drop('Over18', axis = 1, inplace = True)
print(dataset.shape)
```

```
(1470, 31)
```

```
y = dataset.iloc[:, 1]
X = dataset
X.drop('Attrition', axis = 1, inplace = True)
```

Yes No

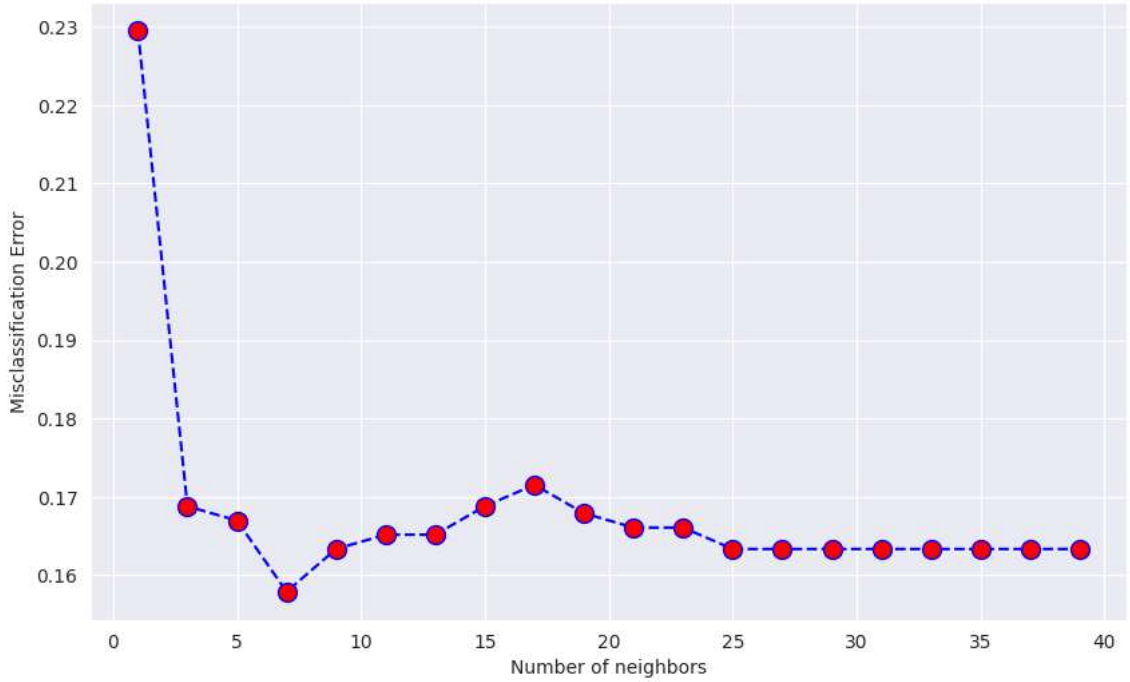
```
from sklearn.preprocessing import LabelEncoder
lb = LabelEncoder()
y = lb.fit_transform(y)


dum_BusinessTravel = pd.get_dummies(dataset['BusinessTravel'],
                                    prefix ='BusinessTravel')
dum_Department = pd.get_dummies(dataset['Department'],
                                prefix ='Department')
dum_EducationField = pd.get_dummies(dataset['EducationField'],
                                    prefix ='EducationField')
dum_Gender = pd.get_dummies(dataset['Gender'],
                            prefix ='Gender', drop_first = True)
dum_JobRole = pd.get_dummies(dataset['JobRole'],
                            prefix ='JobRole')
dum_MaritalStatus = pd.get_dummies(dataset['MaritalStatus'],
                                prefix ='MaritalStatus')
dum_OverTime = pd.get_dummies(dataset['OverTime'],
                            prefix ='OverTime', drop_first = True)
# Adding these dummy variable to input X
X = pd.concat([X, dum_BusinessTravel, dum_Department,
            dum_EducationField, dum_Gender, dum_JobRole,
            dum_MaritalStatus, dum_OverTime], axis = 1)
# Removing the categorical data
X.drop(['BusinessTravel', 'Department', 'EducationField',
        'Gender', 'JobRole', 'MaritalStatus', 'OverTime'],
        axis = 1, inplace = True)

print(X.shape)
print(y.shape)
```

```
(1470, 49)
(1470,)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size = 0.25, random_state = 40)


from sklearn.neighbors import KNeighborsClassifier
neighbors = []
cv_scores = []

from sklearn.model_selection import cross_val_score
# perform 10 fold cross validation
for k in range(1, 40, 2):
    neighbors.append(k)
    knn = KNeighborsClassifier(n_neighbors = k)
    scores = cross_val_score(
```

```
        knn, X_train, y_train, cv = 10, scoring = 'accuracy')
    cv_scores.append(scores.mean())
error_rate = [1-x for x in cv_scores]

# determining the best k
optimal_k = neighbors[error_rate.index(min(error_rate))]
print('The optimal number of neighbors is % d ' % optimal_k)

# plot misclassification error versus k
plt.figure(figsize = (10, 6))
plt.plot(range(1, 40, 2), error_rate, color ='blue', linestyle ='dashed', marker ='o',
        markerfacecolor ='red', markersize = 10)
plt.xlabel('Number of neighbors')
plt.ylabel('Misclassification Error')
plt.show()
```

```
The optimal number of neighbors is  7
```




```
from sklearn.model_selection import cross_val_predict, cross_val_score
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import confusion_matrix

def print_score(clf, X_train, y_train, X_test, y_test, train = True):
    if train:
        print("Train Result:")
        print("------------")
        print("Classification Report: \n {}\n".format(classification_report(
                y_train, clf.predict(X_train))))
        print("Confusion Matrix: \n {}\n".format(confusion_matrix(
                y_train, clf.predict(X_train))))

        res = cross_val_score(clf, X_train, y_train,
                              cv = 10, scoring ='accuracy')
        print("Average Accuracy: \t {0:.4f}".format(np.mean(res)))
        print("Accuracy SD: \t\t {0:.4f}".format(np.std(res)))
        print("accuracy score: {0:.4f}\n".format(accuracy_score(
                y_train, clf.predict(X_train))))
        print("------------------------------------------------------------")

    elif train == False:
        print("Test Result:")
        print("-----------")
        print("Classification Report: \n {}\n".format(
                classification_report(y_test, clf.predict(X_test))))
        print("Confusion Matrix: \n {}\n".format(
                confusion_matrix(y_test, clf.predict(X_test))))
        print("accuracy score: {0:.4f}\n".format(
                accuracy_score(y_test, clf.predict(X_test))))
        print("------------------------------------------------------------")
```

```
knn = KNeighborsClassifier(n_neighbors = 7)
knn.fit(X_train, y_train)
print_score(knn, X_train, y_train, X_test, y_test, train = True)
print_score(knn, X_train, y_train, X_test, y_test, train = False)
```

```
Train Result:
------------
Classification Report:
               precision    recall  f1-score   support

           0       0.86      0.99      0.92       922
           1       0.83      0.19      0.32       180

    accuracy                           0.86      1102
   macro avg       0.85      0.59      0.62      1102
weighted avg       0.86      0.86      0.82      1102


Confusion Matrix:
 [[915   7]
 [145  35]]

Average Accuracy:        0.8421
Accuracy SD:             0.0148
accuracy score: 0.8621

----------------------------------------------------------
Test Result:
-----------
Classification Report:
               precision    recall  f1-score   support

           0       0.84      0.96      0.90       311
           1       0.14      0.04      0.06        57

    accuracy                           0.82       368
   macro avg       0.49      0.50      0.48       368
weighted avg       0.74      0.82      0.77       368


Confusion Matrix:
 [[299  12]
 [ 55   2]]

accuracy score: 0.8179

----------------------------------------------------------
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Separate the target variable (Attrition) from features
X = df.drop('Attrition', axis=1)
y = df['Attrition']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define categorical and numerical columns
categorical_columns = ['Department', 'JobRole', 'MaritalStatus', 'Gender', 'OverTime']
numerical_columns = ['Age', 'DailyRate', 'HourlyRate', 'MonthlyRate', 'NumCompaniesWorked', 'TotalWorkingYears']

# Create transformers for preprocessing
categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

numerical_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

# Combine transformers using a ColumnTransformer
preprocessor = ColumnTransformer(
```

```
        transformers=[
            ('num', numerical_transformer, numerical_columns),
            ('cat', categorical_transformer, categorical_columns)
        ],
        remainder='passthrough'  # Include any columns not specified in transformers
    )

    # Create pipelines for models
    logistic_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                                        ('classifier', LogisticRegression(random_state=42))])

    tree_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                                    ('classifier', DecisionTreeClassifier(random_state=42))])

    from sklearn.preprocessing import OneHotEncoder

    # Create a OneHotEncoder object
    encoder = OneHotEncoder(handle_unknown='ignore')

    import pandas as pd
    from scipy.sparse import csr_matrix

    # Create a sparse matrix
    X = csr_matrix([[1, 2], [3, 4]])

    # Convert the sparse matrix to a dense matrix
    X_dense = X.todense()

    # Create a Pandas DataFrame from the dense matrix
    X_df = pd.DataFrame(X_dense)

    # Print the DataFrame
    print(X_df)
```

```
        0  1
     0  1  2
     1  3  4
```

```
    import pandas as pd
    from scipy.sparse import csr_matrix


    # Convert the sparse matrices to dense matrices
    X_train_dense = X_train.any()
    X_test_dense = X_test.any()

    # Create Pandas DataFrames from the dense matrices
    X_train_df = pd.DataFrame(X_train_dense)
    X_test_df = pd.DataFrame(X_test_dense)


    # Now you can use strings to specify columns
    X_train_encoded = encoder.fit_transform(X_train_df)
    X_test_encoded = encoder.transform(X_test_df)


    # Fit the models
    logistic_pipeline.fit(X_train_encoded, y_train)
    tree_pipeline.fit(X_train_encoded, y_train)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/__init__.py in _get_column_indices(X, key)
    423         try:
--> 424             all_columns = X.columns
    425         except AttributeError:
```

```
# Make predictions
logistic_predictions = logistic_pipeline.predict(X_test_encoded)
tree_predictions = tree_pipeline.predict(X_test_encoded)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-50-91089eb01e09> in <cell line: 3>()
      1 Pipeline: logistic_pipeline
      2 # Make predictions
----> 3 logistic_predictions = logistic_pipeline.predict(X_test_encoded)
      4 tree_predictions = tree_pipeline.predict(X_test_encoded)

4 frames
/usr/local/lib/python3.10/dist-packages/sklearn/compose/_column_transformer.py in _iter(self, fitted, replace_strings,
column_as_strings)
    348
    349                 transformers = [
--> 350                     replace_passthrough(*trans) for trans in self.transformers_
    351                 ]
    352             else:

AttributeError: 'ColumnTransformer' object has no attribute 'transformers_'
```

SEARCH STACK OVERFLOW

```
# Evaluate the models
print("Logistic Regression:")
print("Accuracy:", accuracy_score(y_test, logistic_predictions))
print("Classification Report:\n", classification_report(y_test, logistic_predictions))
print("Confusion Matrix:\n", confusion_matrix(y_test, logistic_predictions))

print("\nDecision Tree:")
print("Accuracy:", accuracy_score(y_test, tree_predictions))
print("Classification Report:\n", classification_report(y_test, tree_predictions))
print("Confusion Matrix:\n", confusion_matrix(y_test, tree_predictions))
```

```
Logistic Regression:
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-51-18a7634c911d> in <cell line: 3>()
      1 # Evaluate the models
      2 print("Logistic Regression:")
----> 3 print("Accuracy:", accuracy_score(y_test, logistic_predictions))
      4 print("Classification Report:\n", classification_report(y_test, logistic_predictions))
      5 print("Confusion Matrix:\n", confusion_matrix(y_test, logistic_predictions))

NameError: name 'logistic_predictions' is not defined
```

SEARCH STACK OVERFLOW

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Encode the categorical features
categorical_features = ['Gender', 'Department']
for feature in categorical_features:
    df = pd.get_dummies(df, drop_first=True, columns=[feature])

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(df[['Age', 'DailyRate', 'Gender', 'Department']], df['Attrition'], test_size=0.25, random

# Train the logistic regression model
from sklearn.linear_model import LogisticRegression
logistic_model = LogisticRegression(random_state=42)
logistic_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = logistic_model.predict(X_test)

# Calculate the performance metrics
```

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1_score = f1_score(y_test, y_pred)

# Print the performance metrics
print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
print('F1 score:', f1_score)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-59-9019eda5ba8f> in <cell line: 7>()
      6 categorical_features = ['Gender', 'Department']
      7 for feature in categorical_features:
----> 8     df = pd.get_dummies(df, drop_first=True, columns=[feature])
      9
     10 # Split the data into train and test sets

                                  3 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in _raise_if_missing(self, key, indexer, axis_name)
   6128                 if use_interval_msg:
   6129                     key = list(key)
-> 6130                 raise KeyError(f"None of [{key}] are in the [{axis_name}]")
   6131
   6132             not_found = list(ensure_index(key)[missing_mask.nonzero()[0]].unique())

KeyError: "None of [Index(['Gender'], dtype='object')] are in the [columns]"
```

SEARCH STACK OVERFLOW