Diabete prediction

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases.

The data is provided as a CSV file, downloaded from https://www.kaggle.com/datasets/mathchi/diabetes-data-set/download (https://www.kaggle.com/datasets/mathchi/diabetes-data-set/download)

The objective is to predict based on diagnostic measurements whether a patient has diabetes.

Explore data

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.metrics import r2_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.feature_selection import RFE
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
diab = pd.read_csv("files/diabetes.csv")
diab.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outc
0	6	148	72	35	0	33.6	0.627	50	
1	1	85	66	29	0	26.6	0.351	31	
2	8	183	64	0	0	23.3	0.672	32	
3	1	89	66	23	94	28.1	0.167	21	
4	0	137	40	35	168	43.1	2.288	33	

In [3]:
```
diab.info()
```

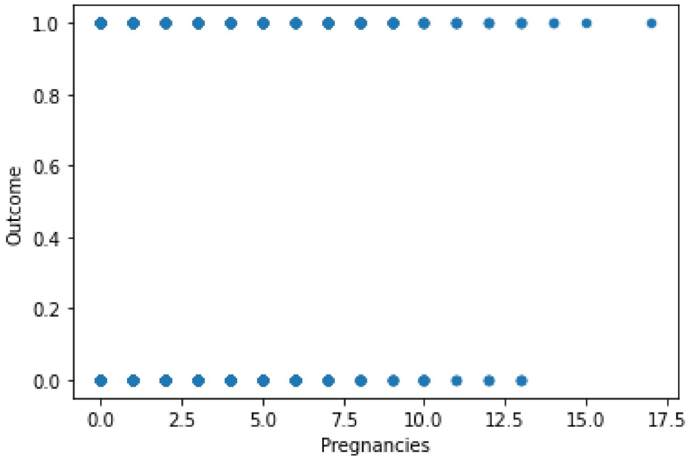
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

From data information, we can see that

- there is no missing data in this dataset
- datatype of all features are numerical

In [4]:
```
# Visualize the data with a scatter plot (x is Pregnancies, y as Outcome)
diab.plot.scatter(x='Pregnancies', y='Outcome')
```

Out[4]: <AxesSubplot:xlabel='Pregnancies', ylabel='Outcome'>

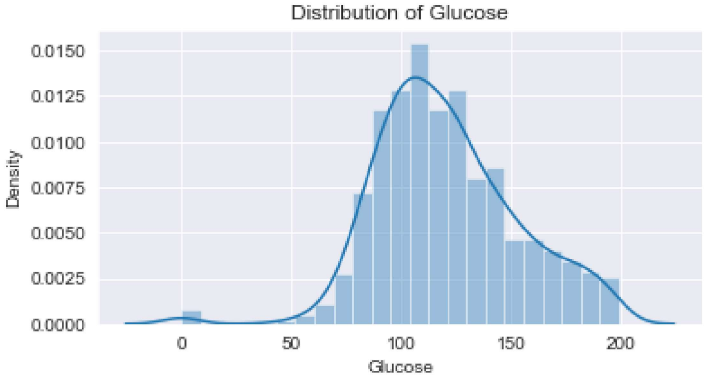


Outliers

visulized data to check outliers

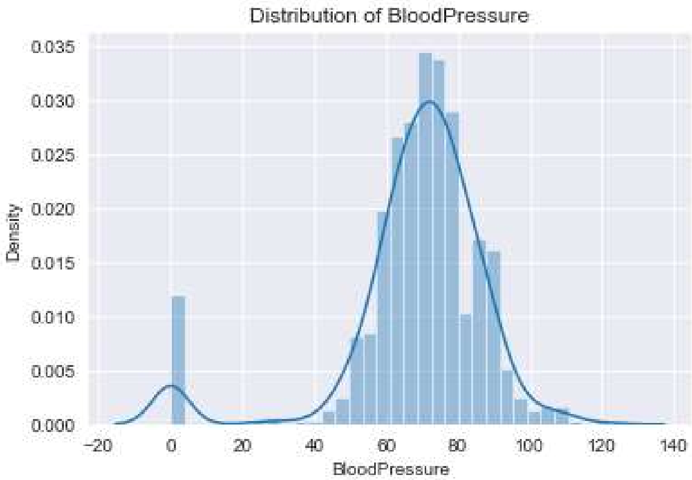
In [5]:
```
plt.figure(figsize=(6,3))
sns.set_style('darkgrid')
sns.distplot(diab.Glucose)
plt.title("Distribution of Glucose")
```

Out[5]: Text(0.5, 1.0, 'Distribution of Glucose')



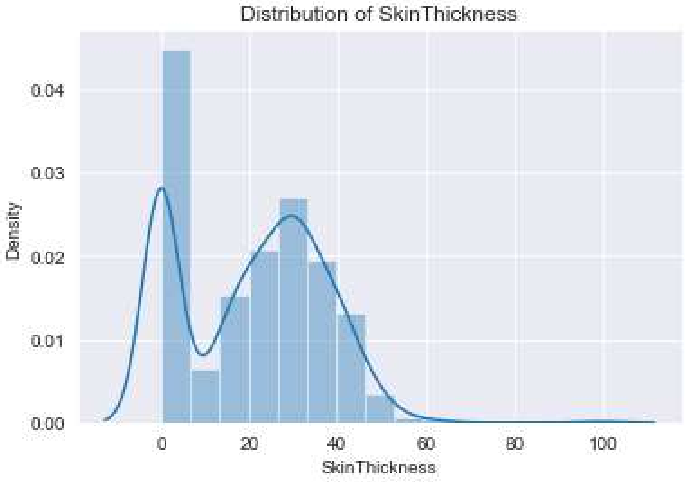
In [6]:
```
sns.distplot(diab.BloodPressure)
plt.title("Distribution of BloodPressure")
```

Out[6]: Text(0.5, 1.0, 'Distribution of BloodPressure')



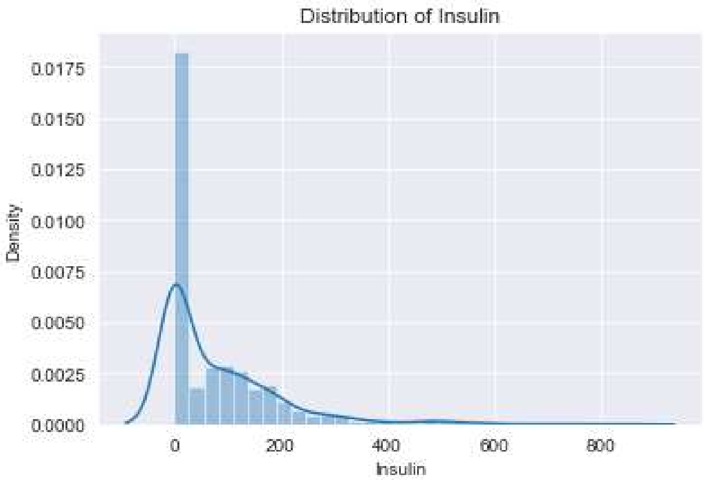
In [7]:
```
sns.distplot(diab.SkinThickness)
plt.title("Distribution of SkinThickness")
```

Out[7]: Text(0.5, 1.0, 'Distribution of SkinThickness')



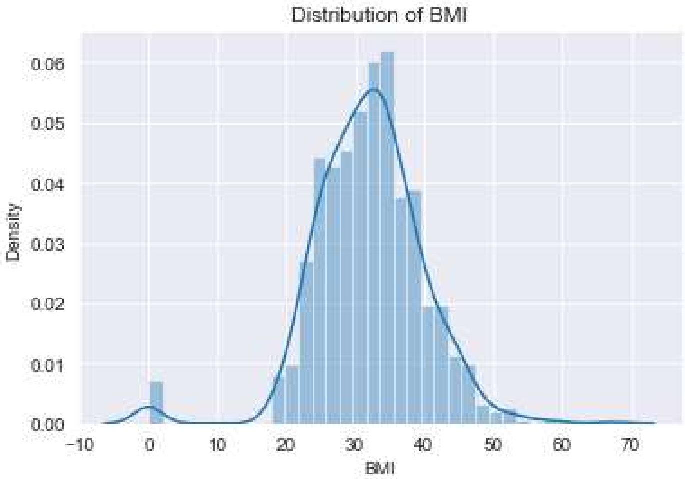
In [8]:
```
sns.distplot(diab.Insulin)
plt.title("Distribution of Insulin")
```

Out[8]: Text(0.5, 1.0, 'Distribution of Insulin')



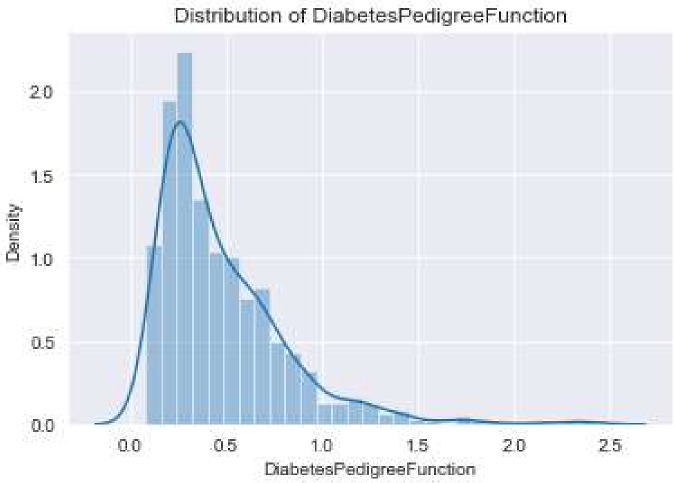
In [9]:
```
sns.distplot(diab.BMI)
plt.title("Distribution of BMI")
```

Out[9]: Text(0.5, 1.0, 'Distribution of BMI')



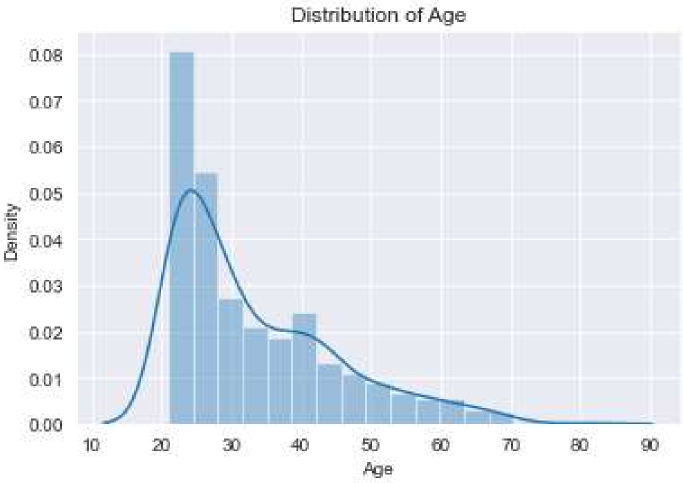
In [10]:
```
sns.distplot(diab.DiabetesPedigreeFunction)
plt.title("Distribution of DiabetesPedigreeFunction")
```

Out[10]: Text(0.5, 1.0, 'Distribution of DiabetesPedigreeFunction')



In [11]:
```
sns.distplot(diab.Age)
plt.title("Distribution of Age")
```

Out[11]: Text(0.5, 1.0, 'Distribution of Age')



Remove outliers

- As can be seen from above plots, there are abnormal data with value=0 which are outliers in features: Glucose, BloodPressure, SkinThickness, Insulin and BMI
- Remove outliers

In [12]:
```
# remove outliers
diab_clean = diab.drop(diab.index[(diab.Glucose==0)|
                                  (diab.BloodPressure==0)|
                                  (diab['Insulin']==0)|
                                  (diab.BMI==0)|
                                  (diab.DiabetesPedigreeFunction==0)]
                      )
diab_clean.info()
```

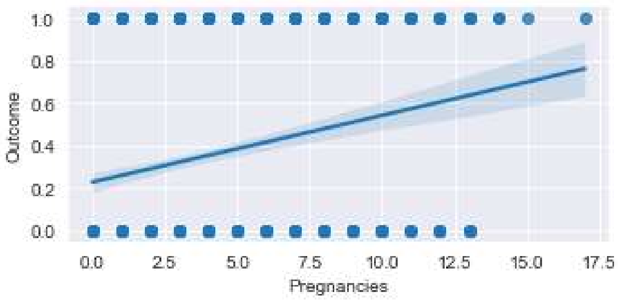
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 392 entries, 3 to 765
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               392 non-null    int64  
 1   Glucose                   392 non-null    int64  
 2   BloodPressure             392 non-null    int64  
 3   SkinThickness             392 non-null    int64  
 4   Insulin                   392 non-null    int64  
 5   BMI                       392 non-null    float64
 6   DiabetesPedigreeFunction  392 non-null    float64
 7   Age                       392 non-null    int64  
 8   Outcome                   392 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 30.6 KB
```

Data Normalisation

- Data normalisation is the organization of data to appear similar across all records and fields
- It increases the cohesion of entry types leading to cleansing, lead generation, segmentation, and higher quality data

In [13]:
```
# The lmplot() function from the Seaborn module is intended for exploring linear relatio
sns.lmplot("Pregnancies", "Outcome", diab, size = 2.5, aspect = 2)
```

Out[13]:
```
<seaborn.axisgrid.FacetGrid at 0x1322d574dc0>
```



Change data type into float

so all values of various features will be min-max normalized between 0 and 1

In [14]:
```
diab_clean=diab_clean.astype('float')
diab_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 392 entries, 3 to 765
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               392 non-null    float64
 1   Glucose                   392 non-null    float64
 2   BloodPressure             392 non-null    float64
 3   SkinThickness             392 non-null    float64
 4   Insulin                   392 non-null    float64
 5   BMI                       392 non-null    float64
 6   DiabetesPedigreeFunction  392 non-null    float64
 7   Age                       392 non-null    float64
 8   Outcome                   392 non-null    float64
dtypes: float64(9)
memory usage: 30.6 KB
```

In [15]:
```
# data normalization: here we do a simple min-max normalization
index = list(diab_clean.index.values)
for feature in diab_clean:
    for i in index:
        diab_clean[feature][i] = (float(diab_clean[feature][i])-float(diab_clean[feature
```

In [16]:
```
diab_clean.head()
```

Out[16]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	
3	0.058824	0.232394	0.488372	0.285714	0.096154	0.202454	0.035118	0.(
4	0.000000	0.691557	0.360798	0.553531	0.198490	0.641242	0.944651	0.‹
6	0.176471	0.393227	0.452750	0.505695	0.103917	0.460369	0.089263	0.:
8	0.117647	0.994944	0.635167	0.712984	0.641803	0.452895	0.051525	0.(
13	0.058824	0.954492	0.543959	0.362187	1.000000	0.446915	0.152159	0.'

Logistic Regression Model

In [17]:
```
diab_clean.describe()
```

Out[17]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
count	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000	392.0
mean	0.240058	0.631035	0.676165	0.514874	0.257825	0.568156	0.2
std	0.235545	0.162738	0.131379	0.198217	0.197605	0.142712	0.1
min	0.000000	0.232394	0.224437	0.108685	0.023294	0.202454	0.0
25%	0.076923	0.504469	0.583980	0.364037	0.124630	0.462578	0.1
50%	0.153846	0.615711	0.659583	0.507086	0.199968	0.556899	0.2
75%	0.357143	0.733273	0.754090	0.635521	0.321859	0.644488	0.3
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0

Training Logistic Regression Model

In [18]:
```
# Split data into training(80%) and testing data (20%) and use random_state=142
train, test = train_test_split(diab_clean, test_size=0.2, random_state=142)
print(train.shape)
print(test.shape)
```

```
(313, 9)
(79, 9)
```

In [19]:
```
# Getting input data and targets for building prediction model
X_train = train.drop(['Outcome'], axis=1)
y_train = train['Outcome']
X_test = test.drop(['Outcome'], axis=1)
y_test = test['Outcome']

print("X_train shape: ", X_train.shape)
print("y_train shape: ", y_train.shape)
print("X_test shape: ", X_test.shape)
print("y_test shape: ", y_test.shape)
```

```
X_train shape:  (313, 8)
y_train shape:  (313,)
X_test shape:  (79, 8)
y_test shape:  (79,)
```

In [20]:
```
# Training Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)
```

Out[20]:
```
LogisticRegression()
```

In [21]:
```
# Doing predictions on train and test set
y_hat_train = model.predict(X_train)
y_hat_test = model.predict(X_test)
```

Evaluation

To evaluate a classification model we want to look at how many cases were correctly classified and how many were in error. In this case we have two outcomes - 0 and 1. SKlearn has some useful tools, the accuracy_score () function gives a score from 0-1 for the proportion correct. The confusion_matrix (http://scikit-learn.org/stable/modules/model_evaluation.html#confusion-matrix) function shows how many were classified correctly and what errors were made. Use these to summarise the performance of your model (these functions have already been imported above).

In [22]:
```
# Evaluate the performance of your trained model
print("Accuracy score on training set: ", accuracy_score(y_train, y_hat_train))
print("Accuracy score on testing set: ", accuracy_score(y_test, y_hat_test))
```

```
Accuracy score on training set:  0.7795527156549521
Accuracy score on testing set:  0.6962025316455697
```

As we can see that model performance is not so good. Also, there is a gap in the accuracy scores for training and testing set, so there might be overfitting of the model.

In [23]:
```
# Checking confusion matrix
print("Confusion matrix on test set: ")
print(confusion_matrix(y_test, y_hat_test))
```

```
Confusion matrix on test set:
[[44  4]
 [20 11]]
```

In [24]:
```
print("Confusion matrix on train set: ")
print(confusion_matrix(y_train, y_hat_train))
```

```
Confusion matrix on train set:
[[199  15]
 [ 54  45]]
```

Feature Selection

Since there might be overfitting in our model, we will select which features to use as input to the classifier to see how models with less features perform?

This process can be automated. The sklearn RFE function (http://scikit-learn.org/stable/modules/feature_selection.html#recursive-feature-elimination) implements **Recursive Feature Estimation** which removes features one by one, evaluating the model each time and selecting the best model for a target number of features. Use RFE to select features for a model with various features

In [25]:
```
# creating RFE object
diab_Reg = LogisticRegression()
rfe = RFE(estimator=diab_Reg, n_features_to_select=5, step=1)
rfe.fit(X_train, y_train)
```

Out[25]:
```
RFE(estimator=LogisticRegression(), n_features_to_select=5)
```

In [26]:
```
# doing evaluation
y_test_hat = rfe.predict(X_test)
print("accuracy score on test set: ", accuracy_score(y_test, y_test_hat))
```

```
accuracy score on test set:  0.6708860759493671
```

In [27]:
```
# summarize all features
for i in range(X_train.shape[1]):
    print('Column: %d, Selected %s, Rank: %.3f' % (i, rfe.support_[i], rfe.ranking_[i]))
```

```
Column: 0, Selected True, Rank: 1.000
Column: 1, Selected True, Rank: 1.000
Column: 2, Selected False, Rank: 4.000
Column: 3, Selected True, Rank: 1.000
Column: 4, Selected False, Rank: 2.000
Column: 5, Selected False, Rank: 3.000
Column: 6, Selected True, Rank: 1.000
Column: 7, Selected True, Rank: 1.000
```

In [28]:
```
# to increment number of features, one at each time
acc_scores = []
for i in range(1,9):
    clf = LogisticRegression()
    rfe = RFE(estimator=clf, n_features_to_select=i)
    # training model
    rfe.fit(X_train, y_train)
    # predicting on test set
    y_pred = rfe.predict(X_test)
    acc_score = accuracy_score(y_test, y_pred)
    # print this
    print("Acc on test set using", i, "features: ", acc_score)
    # append to the list
    acc_scores.append(acc_score)
```

```
Acc on test set using 1 features:  0.6835443037974683
Acc on test set using 2 features:  0.6835443037974683
Acc on test set using 3 features:  0.6962025316455697
Acc on test set using 4 features:  0.6708860759493671
Acc on test set using 5 features:  0.6708860759493671
Acc on test set using 6 features:  0.6962025316455697
Acc on test set using 7 features:  0.6962025316455697
Acc on test set using 8 features:  0.6962025316455697
```

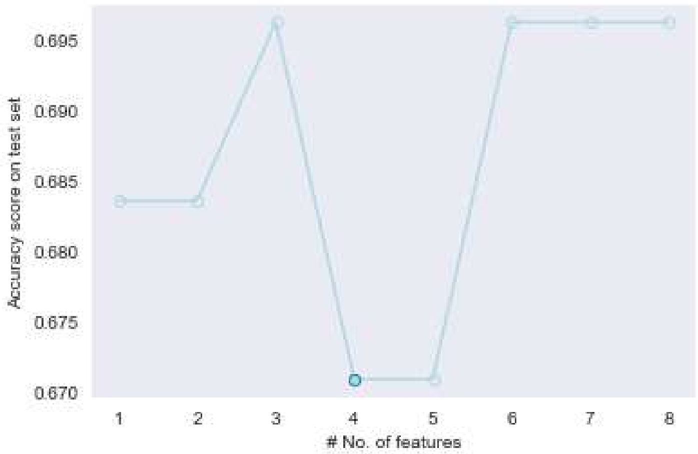
In [29]:
```
# Visulize accuracy score on test set using RFE by using different number of features
estimator = LogisticRegression()
acc_scores = []
for i in range(1, 9):
    selector = RFE(estimator, i)
    selector = selector.fit(X_train, y_train)
    supp = selector.get_support()

    predicted = selector.predict(X_test)
    acc_score = accuracy_score(y_test, predicted)
    acc_scores.append(acc_score)

best = 1
for item in acc_scores:
    if item < acc_scores[best - 1]:
        best = acc_scores.index(item) + 1

plt.grid()
plt.xlabel('# No. of features')
plt.ylabel('Accuracy score on test set')
plt.plot(range(1, 9), acc_scores, marker = 'o', color = 'lightblue', markeredgewidth = 1
plt.plot(best, acc_scores[best-1], marker = 'o', markerfacecolor = 'lightblue')
```

Out[29]:
```
[<matplotlib.lines.Line2D at 0x1322d83e6a0>]
```



Conclusion

Logistic Regression Models perform better in this case when 3 features are selected with accuracy scores close to 0.7 .

Classifying Diabetes with KNN Classifier

Model Training

In [30]:
```python
# Import the KNN classifier
from sklearn.neighbors import KNeighborsClassifier

# Build a KNN classifier model
clf_knn = KNeighborsClassifier(n_neighbors=1)

# Train the model with the training data
clf_knn.fit(X_train, y_train)
```

Out[30]:
```
KNeighborsClassifier(n_neighbors=1)
```

Evaluating Model

In [31]:
```python
from sklearn.metrics import accuracy_score
y_pred = clf_knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy is: %.4f\n" % accuracy)
```

```
Accuracy is: 0.6329
```

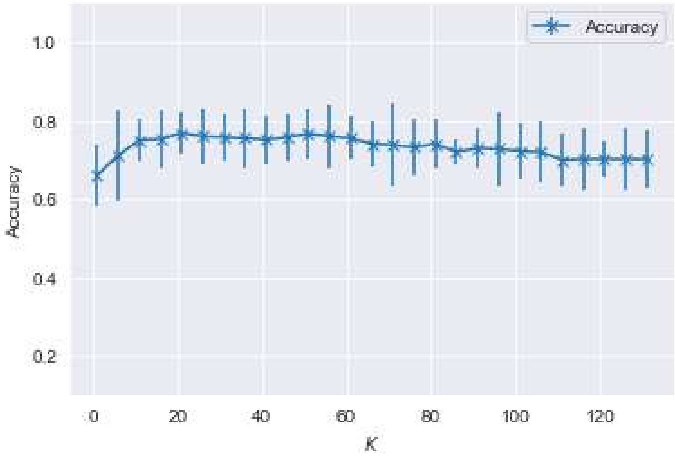
Parameter Tuning with Cross Validation (CV)

Explore a CV method that can be used to tune the hyperparameter K using the above training and test data

In [33]:
```python
X_data = diab_clean.drop(['Outcome'], axis=1)
y_data = diab_clean['Outcome']
```

In [34]:
```
from sklearn.model_selection import cross_val_score, KFold
import matplotlib.pyplot as plt
X_data = diab_clean.drop(['Outcome'], axis=1)
y_data = diab_clean['Outcome']
cv_scores = []
cv_scores_std = []
k_range = range(1, 135, 5)
for i in k_range:
    clf = KNeighborsClassifier(n_neighbors = i)
    scores = cross_val_score(clf, X_data, y_data, scoring='accuracy', cv=KFold(n_splits=
    cv_scores.append(scores.mean())
    cv_scores_std.append(scores.std())

# Plot the relationship
plt.errorbar(k_range, cv_scores, yerr=cv_scores_std, marker='x', label='Accuracy')
plt.ylim([0.1, 1.1])
plt.xlabel('$K$')
plt.ylabel('Accuracy')
plt.legend(loc='best')
plt.show()
```



Find the best K

In [35]:
```
from sklearn.model_selection import GridSearchCV
parameter_grid = {'n_neighbors': range(1, 135, 5)}
knn_clf = KNeighborsClassifier()
gs_knn = GridSearchCV(knn_clf, parameter_grid, scoring='accuracy', cv=KFold(n_splits=10,
gs_knn.fit(X_data, y_data)

print('Best K value: ', gs_knn.best_params_['n_neighbors'])
print('The accuracy: %.4f\n' % gs_knn.best_score_)
```

```
Best K value:  31
The accuracy: 0.7656
```

Conclusion

- With KNN model, the best K value is 31 with accuracy score 0.76.
- KNN model with K=31 performs better than Logistic Regression model in predicting whether a patiant has diabetes.