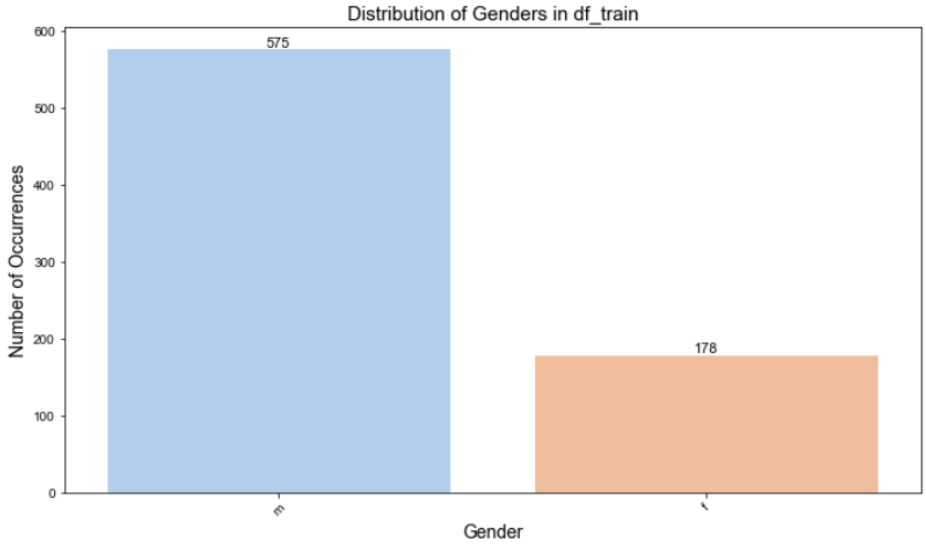
## Bar plot distribution of Genders in df_train

```
gender_counts = df_train['gender'].value_counts()
palette = sns.color_palette("pastel")

plt.figure(figsize=(10, 6))
ax = sns.barplot(x=gender_counts.index, y=gender_counts.values, palette=palette, alpha=0.9)

plt.title('Distribution of Genders in df_train', fontsize=15)
plt.ylabel('Number of Occurrences', fontsize=14)
plt.xlabel('Gender', fontsize=14)

plt.xticks(rotation=45)

for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', fontsize=11, color='black', xytext=(0, 5),
                textcoords='offset points')

# Displaying the grid
sns.set_style("whitegrid")

plt.tight_layout()
plt.show()
```




הסבר התא:

יצרנו דיאגרמת עמודות להבין את התפלגות הסיפורים שיש לנו בסט האימון אשר מסווגות כגבר וכאישה. אפשר לראות שהנתונים לא מאוזנים ויש הרבה יותר סיפורים של גברים מאשר נשים בדאטה, מה שיכול להשפיע על החיזוי הסופי. הצגנו את מספר הסיפורים על ראש כל עמודה (575 של גברים ו 178 של נשים)

## Preprocessing

```
In [12]: df_train_X = list(df_train.loc[:, 'story'])
docs = [re.sub(r'\s+', ' ',
               re.sub(r'^[א-ת]\s+', ' ',
                      re.sub(r'\s+[א-ת]\s+', ' ',
                             re.sub(r'[^א-ת\s]', ' ', str(sen))))) for sen in df_train_X]
encoder = preprocessing.LabelEncoder()
train_y = encoder.fit_transform(df_train.loc[:, 'gender'])
```

הסבר התא:

קודם כל נבצע ניקוי לטקסט האימון. נעבור על כל משפט שיש לנו בנתוני האימון, ונוריד את כל הסימנים המיוחדים דוגמא !@#, את כל האותיות הבודדות (אין מילים בעברית עם אותיות בודדות לכן זה לא רלוונטי לנו לניתוח) כמו א' ב' ג' ד'. נחליף רווחים מרובים ברווח בודד, נגיד יש לנו במשפט משהו כזה:

היי שלום

נחליף את זה ב:

היי שלום

לאחר מכן נבצע קידוד לתווית המטרה שלנו (gender) שזה מין הכותב, נהפוך אותה למשתנה קטגוריאלי אשר יש לו ערך 0/1 במקום m/f. על מנת שמודל למידת המכונה יוכל להתאמן על עמודת המטרה.

תא הבא:

**Create a word level tf-idf**

In [13]:
```
tfidf_vect = TfidfVectorizer(min_df=5, max_df=0.7)
tfidf_vect.fit(docs)
train_X = tfidf_vect.transform(docs)
test_X = tfidf_vect.transform(df_test.loc[:, 'story'])
```

הסבר התא:

נבצע וקטוריזציה למילים שיש לנו בטקסט. התהליך של הוקטוריזציה בעצם לוקח את כל המילים שיש לנו בטקסט ונותן להן משקל על פי כמות הפעמים שהן חוזרות על עצמן. ככה נדע לאלו מילים יש יותר משמעות ולאלו פחות. זהו שלב חשוב בניתוח טקסט.

כעת אחרי שביצענו את ההכנות לנתוני האימון והבדיקה, נבנה מודלי למידת מכונה.

In [14]:
```
train_X, valid_X, train_y, valid_y = model_selection.train_test_split(train_X, train_y, test_size=0.2, random_state=0)
```

חילקנו את הנתונים לאימון 80% ובדיקה 20%.

בחרנו לבנות שלושה מודלים, MLP, Decision Tree, LinearSVC, עבור מודל ה MLP ועץ ההחלטה ביצענו Grid search למציאת הפרמטרים הטובים ביותר. בעצם השיטה הזאת סורקת את כל הפרמטרים של המודל בטווח שהגדרנו, ומחזירה את הפרמטרים שהחזירו את הדיוק הכי גבוה על פי המטריקה שבחרנו שהיא f1-macro.

זה קורה בתא הזה:

**Grid esarch best MLP and DecisionTree parameters**

In [15]:
```
#for MLPClassifier
mlp_params = {
    'hidden_layer_sizes': [(50,), (100,)],
    'activation': ['relu'],
    'solver': ['adam'],
    'alpha': [0.0001],
    'learning_rate': ['constant']
}

mlp_search = GridSearchCV(MLPClassifier(random_state=0), mlp_params, cv=3, scoring='f1_macro', verbose=1, n_jobs=-1)
mlp_search.fit(train_X, train_y)

print("Best parameters for MLPClassifier:")
print(mlp_search.best_params_)

# for DecisionTreeClassifier
tree_params = {
    'criterion': ['gini'],
    'max_depth': [10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 3]
}

tree_search = GridSearchCV(DecisionTreeClassifier(random_state=0), tree_params, cv=3, scoring='f1_macro', verbose=1, n_jobs=-
tree_search.fit(train_X, train_y)

print("\nBest parameters for DecisionTreeClassifier:")
print(tree_search.best_params_)
```

שלבסוף אומר לנו מה הם ההיפר פרמטרים הכי טובים לכל מודל:

Out[15]:

```
GridSearchCV
estimator: MLPClassifier
MLPClassifier
```

```
Best parameters for MLPClassifier:
{'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver': 'adam'}
Fitting 3 folds for each of 8 candidates, totalling 24 fits
```

Out[15]:

```
GridSearchCV
estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

```
Best parameters for DecisionTreeClassifier:
{'criterion': 'gini', 'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 5}
```

לבסוף בנינו את שלושת המודלים, עם ההיפר פרמטרים האופטימליים עבורם:

### Build Decision Tree, LinearSVC and MLPClassifier using best params

In [16]:

```
models = {
    "DecisionTree": DecisionTreeClassifier(random_state=0, **tree_search.best_params_),
    "LinearSVC": LinearSVC(random_state=0),
    "MLPClassifier": MLPClassifier(random_state=0, **mlp_search.best_params_)
}
```

השתמשנו ב random_state=0 שאומר שכל הרצה מחדש של הקוד נקבל את אותן תוצאות.

לאחר מכן אימנו וביצענו חיזוי עבור כל מודל, חישבנו את מדד ה Average F1-Score על פי הדרישות, והצגנו אותו עבור כל מודל, לבסוף שמרנו את המודל הטוב ביותר במשתנה best_model:

### Evaluate each model and select best model by average F1-Score

In [17]:

```
best_f1_avg = 0
best_model_name = ""
best_model = None

for model_name, model in models.items():
    model.fit(train_X, train_y)
    valid_pred = model.predict(valid_X)
    f1_avg = sum(f1_score(valid_pred, valid_y, average=None)) / 2

    if f1_avg > best_f1_avg:
        best_f1_avg = f1_avg
        best_model_name = model_name
        best_model = model

    print(f"{model_name} - Average f1-score: {f1_avg:.3f}")

print(f"\nBest model is {best_model_name} with average F1-score of {best_f1_avg:.3f}")
```

Out[17]:

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=20, min_samples_split=5, random_state=0)
```

```
DecisionTree - Average f1-score: 0.715
```

Out[17]:

```
LinearSVC
LinearSVC(random_state=0)
```

```
LinearSVC - Average f1-score: 0.622
```

Out[17]:

```
MLPClassifier
MLPClassifier(random_state=0)
```

```
MLPClassifier - Average f1-score: 0.617

Best model is DecisionTree with average F1-score of 0.715
```

קיבלנו שהמודל הטוב ביותר שלנו הוא עץ ההחלטה עם Average F1-Score של 0.715.

בתא הבא נבצע חיזוי של נתוני הבדיקה הלא מתוייגים באמצעות המודל הטוב ביותר שלנו שהוא עץ ההחלטה, ונמיר חזרה את ערכי עמודת המטרה ל m ו f לפני שנשמור אותם בקובץ אקסל:

## Predict the test data, and save the results in dataframe.

```
test_pred = tree_classifier.predict(test_X)
labels = ['f','m']
pred = []
for i in range(len(test_pred)):
  pred.append(labels[test_pred[i]])

test_example_id = list(df_test.loc[:,'test_example_id'])
data = {'test_example_id':test_example_id, 'predicted_category':pred}
df_predicted = pd.DataFrame(data,
                    columns=['test_example_id', 'predicted_category'])
df_predicted.head()
```

]:

| | test_example_id | predicted_category |
|---|---|---|
| 0 | 0 | m |
| 1 | 1 | m |
| 2 | 2 | m |
| 3 | 3 | f |
| 4 | 4 | f |

נשמור את כל הנתונים בתוך dataframe, הטבלה תכיל את המספר המזהה של הבדיקה, ואת הקטגוריה שהמודל חזה עבורה (m/f) זכר או נקבה. בתא האחרון בתרגיל נשמור את הנתונים לתוך קובץ אקסל בשם:

Classification_results.csv

### Save output to csv (optional)

After you're done save your output to the 'classification_results.csv' csv file.
We assume that the dataframe with your results contain the following columns:

- column 1 (left column): 'test_example_id' - the same id associated to each of the test stories to be predicted.
- column 2 (right column): 'predicted_category' - the predicted gender value for each of the associated story.

Assuming your predicted values are in the `df_predicted` dataframe, you should save you're results as following:

In [16]:
```
df_predicted.to_csv('classification_results.csv',index=False)
```