

```
In [1]: import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import (
    OneHotEncoder,
    StandardScaler,
    MinMaxScaler
)
from sklearn.impute import SimpleImputer

from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    classification_report, confusion_matrix
)

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

from sklearn.feature_selection import SelectKBest, chi2, RFE

# -----
# 1. Reading dataset
# -----
df = pd.read_csv("bank.csv")
print("Shape:", df.shape)
print("\nColumns:\n", df.columns.tolist())
print("\nHead:\n", df.head())
```

Shape: (11162, 17)

Columns:

['age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays', 'previous', 'poutcome', 'deposit']

Head:

	age	job	marital	education	default	balance	housing	loan	contact	\
0	59	admin.	married	secondary	no	2343	yes	no	unknown	
1	56	admin.	married	secondary	no	45	no	no	unknown	
2	41	technician	married	secondary	no	1270	yes	no	unknown	
3	55	services	married	secondary	no	2476	yes	no	unknown	
4	54	admin.	married	tertiary	no	184	no	no	unknown	

	day	month	duration	campaign	pdays	previous	poutcome	deposit
0	5	may	1042	1	-1	0	unknown	yes
1	5	may	1467	1	-1	0	unknown	yes
2	5	may	1389	1	-1	0	unknown	yes
3	5	may	579	1	-1	0	unknown	yes
4	5	may	673	2	-1	0	unknown	yes

```
In [2]: # -----
# 2. Handle missing values
# -----
# Missing values overview
missing = df.isnull().sum().sort_values(ascending=False)
print("\nMissing values per column:\n", missing)

missing_markers = ["unknown", "UNKNOWN", "?", "NA", "N/A", "null", "None", ""]
df = df.replace(missing_markers, np.nan)

missing2 = df.isnull().sum().sort_values(ascending=False)
print("\nMissing values after replacing markers:\n", missing2)
```

Missing values per column:

```
age      0
job      0
marital  0
education 0
default  0
balance  0
housing  0
loan     0
contact  0
day      0
month    0
duration 0
campaign 0
pdays   0
previous 0
poutcome 0
deposit  0
dtype: int64
```

Missing values after replacing markers:

```
poutcome 8326
contact  2346
education 497
job       70
age       0
default   0
balance   0
housing   0
marital   0
loan      0
day       0
duration  0
month     0
campaign  0
pdays    0
previous  0
deposit   0
dtype: int64
```

```
In [3]: # -----
# 3. Define target + Train/Test split + Baseline Models + GridSearch
# -----

target_col = "y" if "y" in df.columns else (df.columns[-1] if "deposit" not in df.columns else None)
print("\nTarget column:", target_col)

X = df.drop(columns=[target_col])
y = df[target_col].copy()

if y.dtype == "object" or str(y.dtype).startswith("category"):
    y_unique = sorted([str(v).lower() for v in y.dropna().unique()])
    if set(y_unique) <= {"yes", "no"}:
        y = y.astype(str).str.lower().map({"no": 0, "yes": 1})
    else:
        y, _ = pd.factorize(y)

num_cols = X.select_dtypes(include=["number"]).columns.tolist()
cat_cols = [c for c in X.columns if c not in num_cols]
```

```

print("\n#Numeric columns:", len(num_cols))
print("#Categorical columns:", len(cat_cols))

# Preprocessing (FULL models)
numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore", sparse_output=False))
])

preprocess = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, num_cols),
        ("cat", categorical_transformer, cat_cols)
    ],
    remainder="drop"
)

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y if len(np.unique(y)) > 1 else None
)

models = {
    "Logistic Regression": LogisticRegression(max_iter=2000),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "SVM (RBF)": SVC(probability=True)
}

param_grids = {
    "Logistic Regression": {
        "clf__C": [0.1, 1, 10],
        "clf__solver": ["lbfgs"]
    },
    "Decision Tree": {
        "clf__max_depth": [None, 5, 10, 20],
        "clf__min_samples_split": [2, 5, 10]
    },
    "Random Forest": {
        "clf__n_estimators": [100, 200],
        "clf__max_depth": [None, 10, 20],
        "clf__min_samples_split": [2, 5]
    },
    "SVM (RBF)": {
        "clf__C": [0.1, 1, 10],
        "clf__gamma": ["scale", "auto"]
    }
}

def evaluate_classifier(name, fitted_model, X_te, y_te):
    y_pred = fitted_model.predict(X_te)
    return {

```

```

        "Model": name,
        "Accuracy": accuracy_score(y_te, y_pred),
        "Precision": precision_score(y_te, y_pred, average="weighted", zero_divi
        "Recall": recall_score(y_te, y_pred, average="weighted", zero_division=0
        "F1": f1_score(y_te, y_pred, average="weighted", zero_division=0)
    }

print("\n=====")
print("3) Grid Search on FULL feature set")
print("=====")

grid_results = []
best_estimators = {}

for name, clf in models.items():
    print(f"\nStarting GridSearch for: {name}")

    pipe = Pipeline(steps=[
        ("preprocess", preprocess),
        ("clf", clf)
    ])

    grid = GridSearchCV(
        estimator=pipe,
        param_grid=param_grids[name],
        scoring="f1_weighted",
        cv=5,
        n_jobs=-1,
        verbose=1 # helps you see progress
    )

    grid.fit(X_train, y_train)

    print(f"Finished GridSearch for: {name}")
    best_estimators[name] = grid.best_estimator_

    metrics = evaluate_classifier(name, grid.best_estimator_, X_test, y_test)
    metrics["BestParams"] = grid.best_params_
    grid_results.append(metrics)

results_full = pd.DataFrame(grid_results).sort_values(by="F1", ascending=False)

print("\nModel Performance (FULL features) - sorted by F1:\n")
print(results_full.to_string(index=False))

def get_feature_names_from_column_transformer(ct: ColumnTransformer):
    feature_names = []
    for name, trans, cols in ct.transformers_:
        if name == "remainder" and trans == "drop":
            continue

        if hasattr(trans, "named_steps"):
            last = list(trans.named_steps.values())[-1]
        else:
            last = trans

        if hasattr(last, "get_feature_names_out"):
            try:

```

```

        fn = last.get_feature_names_out(cols)
    except Exception:
        fn = last.get_feature_names_out()
        feature_names.extend(fn.tolist())
    else:
        if isinstance(cols, list):
            feature_names.extend(cols)
        else:
            feature_names.append(cols)

    return feature_names

```

Target column: deposit

#Numeric columns: 7

#Categorical columns: 9

=====

3) Grid Search on FULL feature set

=====

Starting GridSearch for: Logistic Regression

Fitting 5 folds for each of 3 candidates, totalling 15 fits

Finished GridSearch for: Logistic Regression

Starting GridSearch for: Decision Tree

Fitting 5 folds for each of 12 candidates, totalling 60 fits

Finished GridSearch for: Decision Tree

Starting GridSearch for: Random Forest

Fitting 5 folds for each of 12 candidates, totalling 60 fits

Finished GridSearch for: Random Forest

Starting GridSearch for: SVM (RBF)

Fitting 5 folds for each of 6 candidates, totalling 30 fits

Finished GridSearch for: SVM (RBF)

Model Performance (FULL features) - sorted by F1:

	Model	Accuracy	Precision	Recall	F1
BestParams					
	Random Forest	0.845052	0.847303	0.845052	0.845146
{ 'clf__max_depth': None, 'clf__min_samples_split': 5, 'clf__n_estimators': 100 }					
	SVM (RBF)	0.836543	0.837935	0.836543	0.836655
{ 'clf__C': 10, 'clf__gamma': 'auto' }					
	Logistic Regression	0.813704	0.814089	0.813704	0.813304
{ 'clf__C': 0.1, 'clf__solver': 'lbfgs' }					
	Decision Tree	0.810121	0.810057	0.810121	0.809979
{ 'clf__max_depth': 10, 'clf__min_samples_split': 5 }					

```

In [4]: # -----
# 4) Feature Selection Method 1: Chi-Square (Filter) + chart

# -----
print("\n=====")
print("4) Feature Selection Method 1: Chi-Square (SelectKBest)")
print("=====")

numeric_transformer_chi2 = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),

```

```

        ("minmax", MinMaxScaler())
    ])

categorical_transformer_chi2 = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore", sparse_output=False))
])

preprocess_chi2 = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer_chi2, num_cols),
        ("cat", categorical_transformer_chi2, cat_cols)
    ],
    remainder="drop"
)

X_train_chi = preprocess_chi2.fit_transform(X_train)
X_test_chi = preprocess_chi2.transform(X_test)

all_features_chi = np.array(get_feature_names_from_column_transformer(preprocess_chi2))

k_chi = min(20, X_train_chi.shape[1])
selector_chi = SelectKBest(score_func=chi2, k=k_chi)

X_train_chi_k = selector_chi.fit_transform(X_train_chi, y_train)
X_test_chi_k = selector_chi.transform(X_test_chi)

selected_mask_chi = selector_chi.get_support()
selected_features_chi = all_features_chi[selected_mask_chi]

print(f"Selected top-{k_chi} features by Chi-Square:\n", selected_features_chi)

# Chart 1: Chi-Square feature scores (bar chart)
chi2_scores_selected = selector_chi.scores_[selected_mask_chi]
order = np.argsort(chi2_scores_selected)[::-1]

plt.figure(figsize=(11, 5))
plt.bar(range(len(selected_features_chi)), chi2_scores_selected[order])
plt.xticks(range(len(selected_features_chi)), selected_features_chi[order], rotation=45)
plt.title("Top Features by Chi-Square Score (SelectKBest)")
plt.ylabel("Chi-Square Score")
plt.tight_layout()
plt.show()

# Evaluate models on Chi2-selected features
models_fs = {
    "Logistic Regression": LogisticRegression(max_iter=2000),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "SVM (RBF)": SVC()
}

results_chi = []
for name, clf in models_fs.items():
    clf.fit(X_train_chi_k, y_train)
    y_pred = clf.predict(X_test_chi_k)
    results_chi.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred, average="weighted", zero_divisor=1e-10)
    })

```

```

    "Recall": recall_score(y_test, y_pred, average="weighted", zero_division=0)
    "F1": f1_score(y_test, y_pred, average="weighted", zero_division=0)
})

```

```

results_chi_df = pd.DataFrame(results_chi).sort_values(by="F1", ascending=False)
print("\nModel Performance (Chi-Square selected features):\n")
print(results_chi_df.to_string(index=False))

```

=====

4) Feature Selection Method 1: Chi-Square (SelectKBest)

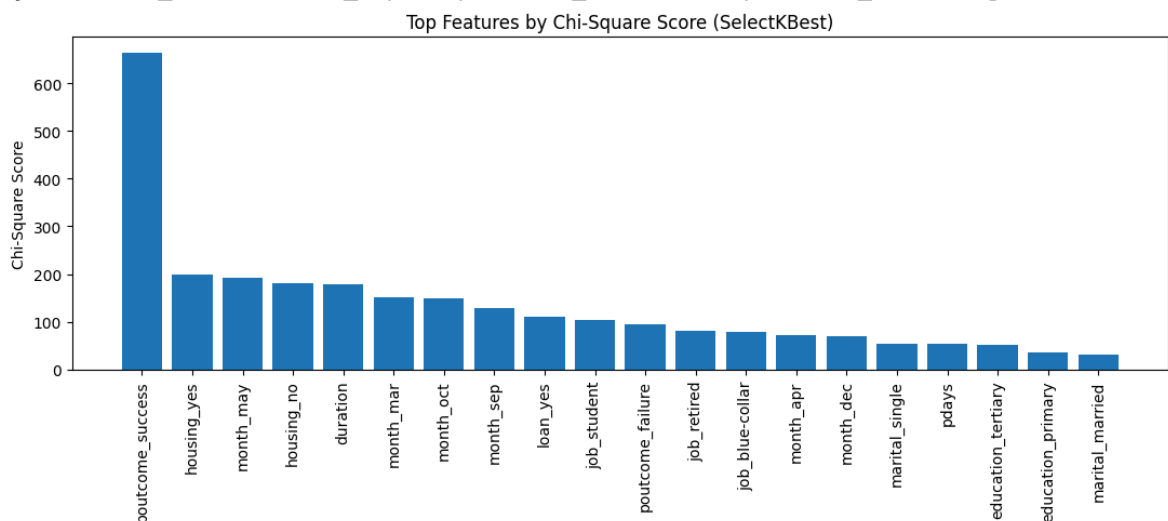
=====

Selected top-20 features by Chi-Square:

```

['duration', 'pdays', 'job_blue-collar', 'job_retired', 'job_student', 'marital_married', 'marital_single', 'education_primary', 'education_tertiary', 'housing_no', 'housing_yes', 'loan_yes', 'month_apr', 'month_dec', 'month_mar', 'month_may', 'month_oct', 'month_sep', 'poutcome_failure', 'poutcome_success']

```



Model Performance (Chi-Square selected features):

	Model	Accuracy	Precision	Recall	F1
	SVM (RBF)	0.817734	0.818478	0.817734	0.817223
	Logistic Regression	0.812360	0.813510	0.812360	0.811687
	Random Forest	0.783251	0.784022	0.783251	0.783385
	Decision Tree	0.759516	0.759362	0.759516	0.759328

```

In [6]: # -----
# 5) Feature Selection Method 2: RFE (Wrapper) + chart :contentReference[oaicite
# -----
print("\n=====")
print("5) Feature Selection Method 2: RFE (Wrapper)")
print("=====")

X_train_std = preprocess.fit_transform(X_train)
X_test_std = preprocess.transform(X_test)

all_features_std = np.array(get_feature_names_from_column_transformer(preprocess

rfe_estimator = LogisticRegression(max_iter=2000)
k_rfe = min(20, X_train_std.shape[1])

rfe = RFE(estimator=rfe_estimator, n_features_to_select=k_rfe, step=1)
rfe.fit(X_train_std, y_train)

X_train_rfe = rfe.transform(X_train_std)
X_test_rfe = rfe.transform(X_test_std)

```

```

selected_features_rfe = all_features_std[rfe.get_support()]
print(f"Selected top-{k_rfe} features by RFE:\n", selected_features_rfe.tolist())

# Chart 2: RFE selected features importance (|LogReg Coef|)

coef_selected = np.abs(rfe.estimator_.coef_)

# If binary classification -> shape (1, k_rfe)
# If multiclass -> shape (n_classes, k_rfe). Use mean magnitude across classes.
if coef_selected.ndim == 2 and coef_selected.shape[0] > 1:
    coef_selected = coef_selected.mean(axis=0)
else:
    coef_selected = coef_selected.ravel()

order = np.argsort(coef_selected)[::-1]

plt.figure(figsize=(11, 5))
plt.bar(range(len(selected_features_rfe)), coef_selected[order])
plt.xticks(range(len(selected_features_rfe)), selected_features_rfe[order], rotate=45)
plt.title("RFE Selected Features Importance (|LogReg Coef|)")
plt.ylabel("Absolute Coefficient Magnitude")
plt.tight_layout()
plt.show()

# Evaluate models on RFE-selected features :contentReference[oaicite:6]{index=6}
results_rfe = []
for name, clf in models_fs.items():
    clf.fit(X_train_rfe, y_train)
    y_pred = clf.predict(X_test_rfe)
    results_rfe.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred, average="weighted", zero_division=0),
        "Recall": recall_score(y_test, y_pred, average="weighted", zero_division=0),
        "F1": f1_score(y_test, y_pred, average="weighted", zero_division=0)
    })

results_rfe_df = pd.DataFrame(results_rfe).sort_values(by="F1", ascending=False)
print("\nModel Performance (RFE selected features):\n")
print(results_rfe_df.to_string(index=False))

```

=====

5) Feature Selection Method 2: RFE (Wrapper)

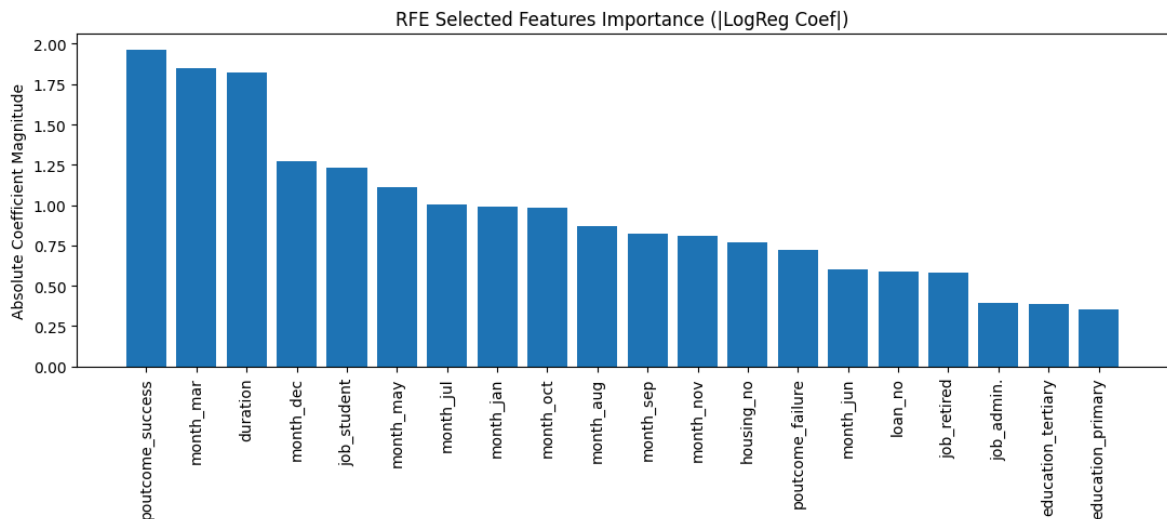
=====

Selected top-20 features by RFE:

```

['duration', 'job_admin.', 'job_retired', 'job_student', 'education_primary', 'education_tertiary', 'housing_no', 'loan_no', 'month_aug', 'month_dec', 'month_jan', 'month_jul', 'month_jun', 'month_mar', 'month_may', 'month_nov', 'month_oct', 'month_sep', 'poutcome_failure', 'poutcome_success']

```



Model Performance (RFE selected features):

	Model	Accuracy	Precision	Recall	F1
	SVM (RBF)	0.823556	0.824909	0.823556	0.823677
Logistic Regression		0.811464	0.812160	0.811464	0.810936
Random Forest		0.775638	0.775998	0.775638	0.775736
Decision Tree		0.737573	0.737386	0.737573	0.737394

```
In [7]: # -----
# 6) Summary + Chart 3: Compare FULL vs Chi2 vs RFE (F1)
# -----
print("\n=====")
print("(6) Final Summary (Best per setting)")
print("=====")

best_full = results_full.iloc[0][["Model", "Accuracy", "Precision", "Recall", "F1"]]
best_chi = results_chi_df.iloc[0][["Model", "Accuracy", "Precision", "Recall", "F1"]]
best_rfe = results_rfe_df.iloc[0][["Model", "Accuracy", "Precision", "Recall", "F1"]]

summary = pd.DataFrame([
    {"Setting": "FULL Features (GridSearch best)", **best_full.to_dict()},
    {"Setting": "Chi-Square Selected Features", **best_chi.to_dict()},
    {"Setting": "RFE Selected Features", **best_rfe.to_dict()},
])

print(summary.to_string(index=False))

# Chart 3: F1 comparison across settings (bar chart) :contentReference[oaicite:7]
full_f1 = results_full[["Model", "F1"]].copy()
full_f1["Setting"] = "FULL"

chi_f1 = results_chi_df[["Model", "F1"]].copy()
chi_f1["Setting"] = "Chi2"

rfe_f1 = results_rfe_df[["Model", "F1"]].copy()
rfe_f1["Setting"] = "RFE"

comp = pd.concat([full_f1, chi_f1, rfe_f1], axis=0)
pivot = comp.pivot(index="Model", columns="Setting", values="F1").fillna(0)

plt.figure(figsize=(10, 5))
pivot.plot(kind="bar")
plt.title("F1 Comparison Across Feature Selection Settings")
plt.ylabel("F1 (weighted)")
```

```
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()

# Optional detailed report for best FULL-features model
best_model_name = results_full.iloc[0]["Model"]
print("\n=====")
print(f"Classification Report (Best FULL-features model): {best_model_name}")
print("=====")

best_pipe = best_estimators[best_model_name]
y_pred_best = best_pipe.predict(X_test)

print(classification_report(y_test, y_pred_best, zero_division=0))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_best))
```

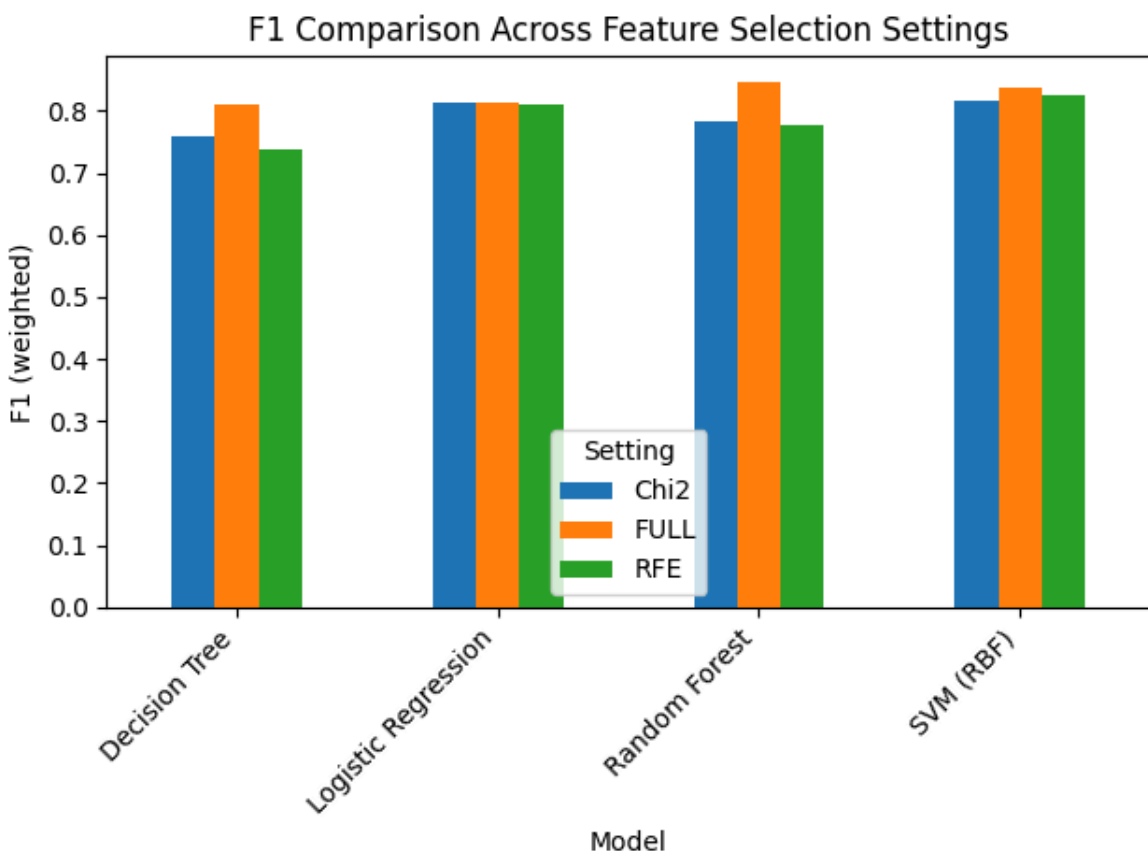
=====

6) Final Summary (Best per setting)

=====

	Setting	Model	Accuracy	Precision	Recall
F1					
FULL Features (GridSearch best)	Random Forest	0.845052	0.847303	0.845052	0.845146
Chi-Square Selected Features	SVM (RBF)	0.817734	0.818478	0.817734	0.817223
RFE Selected Features	SVM (RBF)	0.823556	0.824909	0.823556	0.823677

<Figure size 1000x500 with 0 Axes>



```
=====  
Classification Report (Best FULL-features model): Random Forest  
=====  
              precision    recall  f1-score   support  
  
    0           0.88       0.82      0.85        1175  
    1           0.81       0.87      0.84        1058  
  
   accuracy                0.85        2233  
  macro avg           0.85       0.85      0.85        2233  
weighted avg           0.85       0.85      0.85        2233  
  
Confusion Matrix:  
[[963 212]  
 [134 924]]
```

```
In [ ]:
```