```
from google.colab import files
uploaded = files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving tested.csv to tested.csv

```
import pandas as pd

df = pd.read_csv('tested.csv')
df.head()
```

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 892 | 0 | 3 | Kelly, Mr. James | male | 34.5 | 0 | 0 | 330911 | 7.8292 | NaN | Q |
| **1** | 893 | 1 | 3 | Wilkes, Mrs. James (Ellen Needs) | female | 47.0 | 1 | 0 | 363272 | 7.0000 | NaN | S |
| **2** | 894 | 0 | 2 | Myles, Mr. Thomas Francis | male | 62.0 | 0 | 0 | 240276 | 9.6875 | NaN | Q |
| **3** | 895 | 0 | 3 | Wirz, Mr. Albert | male | 27.0 | 0 | 0 | 315154 | 8.6625 | NaN | S |
| **4** | 896 | 1 | 3 | Hirvonen, Mrs. Alexander (Helga E Lindqvist) | female | 22.0 | 1 | 1 | 3101298 | 12.2875 | NaN | S |

```
# ----------------------------------------------------
# 1. Setup
# ----------------------------------------------------
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use("default")          # keep plain Matplotlib style

# ----------------------------------------------------
# 2. Load the Titanic dataset
#    (replace 'url' with the path to your own CSV if needed)
# ----------------------------------------------------
url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"
df = pd.read_csv(url)

# ----------------------------------------------------
# 3. Quick peek at the data
# ----------------------------------------------------
print("Shape:", df.shape)
df.head()
```

```
Shape: (891, 12)
```

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 | 0 | A/5 21171 | 7.2500 | NaN | S |
| **1** | 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th... | female | 38.0 | 1 | 0 | PC 17599 | 71.2833 | C85 | C |
| **2** | 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26.0 | 0 | 0 | STON/O2. 3101282 | 7.9250 | NaN | S |
| **3** | 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35.0 | 1 | 0 | 113803 | 53.1000 | C123 | S |
| **4** | 5 | 0 | 3 | Allen, Mr. William Henry | male | 35.0 | 0 | 0 | 373450 | 8.0500 | NaN | S |

```
df.head(3)    # first 3 rows
```

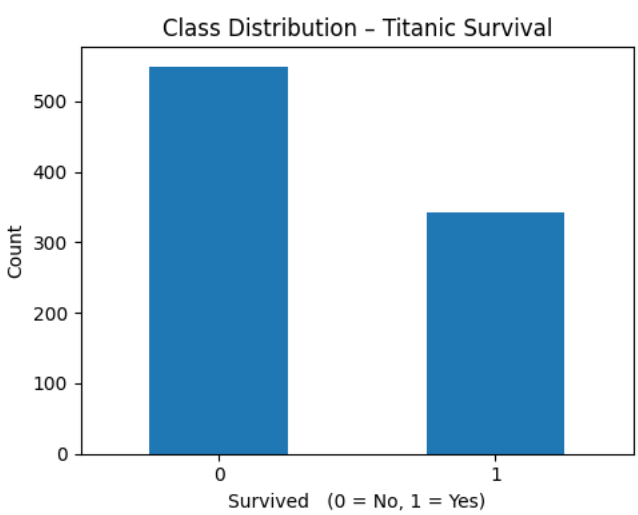
| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 | 0 | A/5 21171 | 7.2500 | NaN | S |
| **1** | 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th... | female | 38.0 | 1 | 0 | PC 17599 | 71.2833 | C85 | C |
| **2** | 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26.0 | 0 | 0 | STON/O2. 3101282 | 7.9250 | NaN | S |

```
missing = df.isna().sum().sort_values(ascending=False)
print("Missing values per column:\n", missing)

# --- Simple handling strategy (feel free to improve) ----
# Numerical columns → fill with median
num_cols = df.select_dtypes(include='number').columns
```

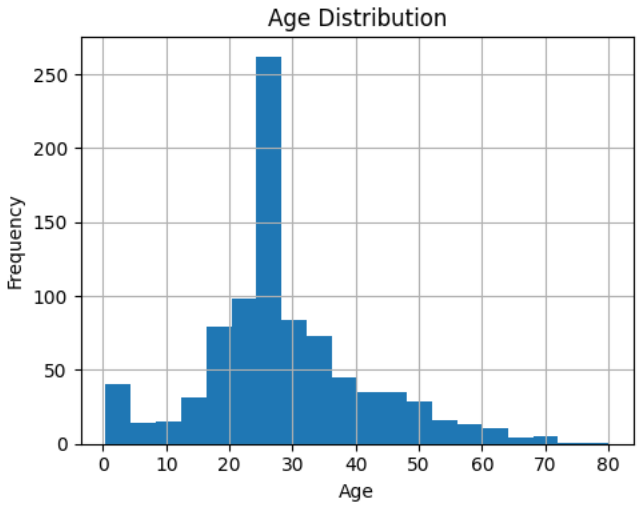
```
df[num_cols] = df[num_cols].fillna(df[num_cols].median())

# Categorical columns → fill with mode
cat_cols = df.select_dtypes(include='object').columns
for col in cat_cols:
    df[col].fillna(df[col].mode()[0], inplace=True)
```

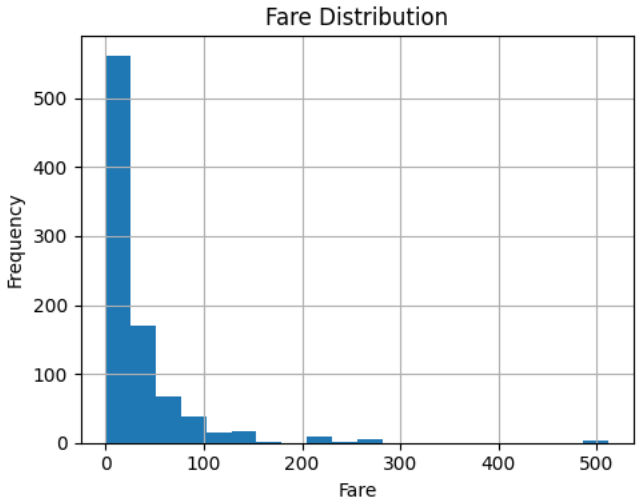
```
Missing values per column:
 Cabin          687
Age             177
Embarked          2
PassengerId       0
Name              0
Pclass            0
Survived          0
Sex               0
Parch             0
SibSp             0
Fare              0
Ticket            0
dtype: int64
/tmp/ipython-input-6-4084604473.py:12: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series thr
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[c


  df[col].fillna(df[col].mode()[0], inplace=True)
```

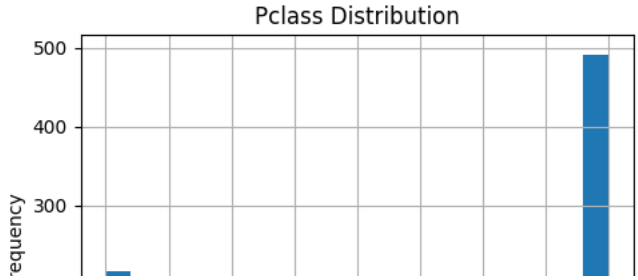
```
plt.figure(figsize=(5,4))
df['Survived'].value_counts().sort_index().plot(kind="bar")
plt.title("Class Distribution – Titanic Survival")
plt.xlabel("Survived   (0 = No, 1 = Yes)")
plt.ylabel("Count")
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```




```
numeric_cols = ['Age', 'Fare', 'Pclass', 'SibSp', 'Parch']

for col in numeric_cols:
    plt.figure(figsize=(5,4))
    df[col].hist(bins=20)
    plt.title(f"{col} Distribution")
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.tight_layout()
    plt.show()
```



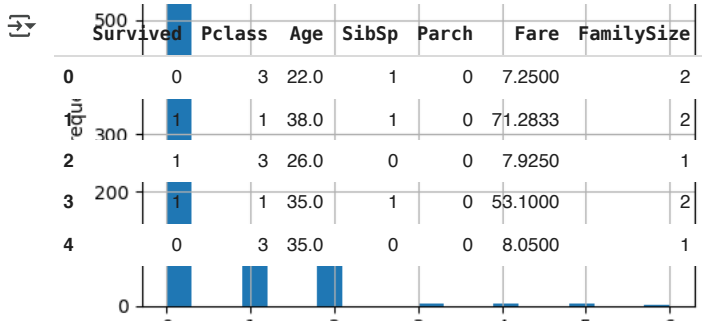




```
# Create new feature: Family Size
df['FamilySize'] = df['SibSp'] + df['Parch'] + 1

# Extract Title from Name (e.g., Mr, Miss, Mrs)
df['Title'] = df['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)

# Group rare titles
df['Title'] = df['Title'].replace(['Lady', 'Countess','Capt','Col', 'Don', 'Dr',
                                   'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')
df['Title'] = df['Title'].replace({'Mlle': 'Miss', 'Ms': 'Miss', 'Mme': 'Mrs'})

# Drop unused columns
df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)

df.head()
```

| | Survived | Pclass | Sex | Age | SibSp | Parch | Fare | Embarked | FamilySize | Title |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 3 | male | 22.0 | 1 | 0 | 7.2500 | S | 2 | Mr |
| 1 | 1 | 1 | female | 38.0 | 1 | 0 | 71.2833 | C | 2 | Mrs |
| 2 | 1 | 3 | female | 26.0 | 0 | 0 | 7.9250 | S | 1 | Miss |
| 3 | 1 | 1 | female | 35.0 | 1 | 0 | 53.1000 | S | 2 | Mrs |
| 4 | 0 | 3 | male | 35.0 | 0 | 0 | 8.0500 | S | 1 | Mr |

```
# One-hot encode categorical columns
df = pd.get_dummies(df, columns=['Sex', 'Embarked', 'Title'], drop_first=True)

df.head()
```



| | Survived | Pclass | Age | SibSp | Parch | Fare | FamilySize | Sex_male | Embarked_Q | Embarked_S | Title_Miss | Title_Mr | Title_M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 3 | 22.0 | 1 | 0 | 7.2500 | 2 | True | False | True | False | True | Fal |
| 1 | 1 | 1 | 38.0 | 1 | 0 | 71.2833 | 2 | False | False | False | False | False | Tr |
| 2 | 1 | 3 | 26.0 | 0 | 0 | 7.9250 | 1 | False | False | True | True | False | Fal |
| 3 | 1 | 1 | 35.0 | 1 | 0 | 53.1000 | 2 | False | False | True | False | False | Tr |
| 4 | 0 | 3 | 35.0 | 0 | 0 | 8.0500 | 1 | True | False | True | False | True | Fal |

```
from sklearn.preprocessing import StandardScaler

# Identify numerical columns (excluding 'Survived')
num_cols = ['Age', 'Fare', 'Pclass', 'SibSp', 'Parch', 'FamilySize']

# Initialize scaler
scaler = StandardScaler()

# Apply scaling
df[num_cols] = scaler.fit_transform(df[num_cols])

df.head()
```

| | Survived | Pclass | Age | SibSp | Parch | Fare | FamilySize | Sex_male | Embarked_Q | Embarked_S | Title_Miss | Title |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0.827377 | -0.565736 | 0.432793 | -0.473674 | -0.502445 | 0.059160 | True | False | True | False | |
| 1 | 1 | -1.566107 | 0.663861 | 0.432793 | -0.473674 | 0.786845 | 0.059160 | False | False | False | False | F |
| 2 | 1 | 0.827377 | -0.258337 | -0.474545 | -0.473674 | -0.488854 | -0.560975 | False | False | True | True | F |
| 3 | 1 | -1.566107 | 0.433312 | 0.432793 | -0.473674 | 0.420730 | 0.059160 | False | False | True | False | F |
| 4 | 0 | 0.827377 | 0.433312 | -0.474545 | -0.473674 | -0.486337 | -0.560975 | True | False | True | False | |

```
# ------------------------------------------------------
# 1. Train-Test Split  (80 % train / 20 % test)
# ------------------------------------------------------
from sklearn.model_selection import train_test_split

X = df.drop('Survived', axis=1)
y = df['Survived']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=42, stratify=y)

print(f"Train shape: {X_train.shape},  Test shape: {X_test.shape}")
```

```
Train shape: (712, 13),  Test shape: (179, 13)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier



# --- Logistic Regression ---
log_reg = LogisticRegression(max_iter=1000, n_jobs=-1)
log_reg.fit(X_train, y_train)

# --- Random Forest ---
rf = RandomForestClassifier(
        n_estimators=200,
        max_depth=None,
        min_samples_split=2,
        min_samples_leaf=1,
        random_state=42,
        n_jobs=-1
    )
rf.fit(X_train, y_train)
```

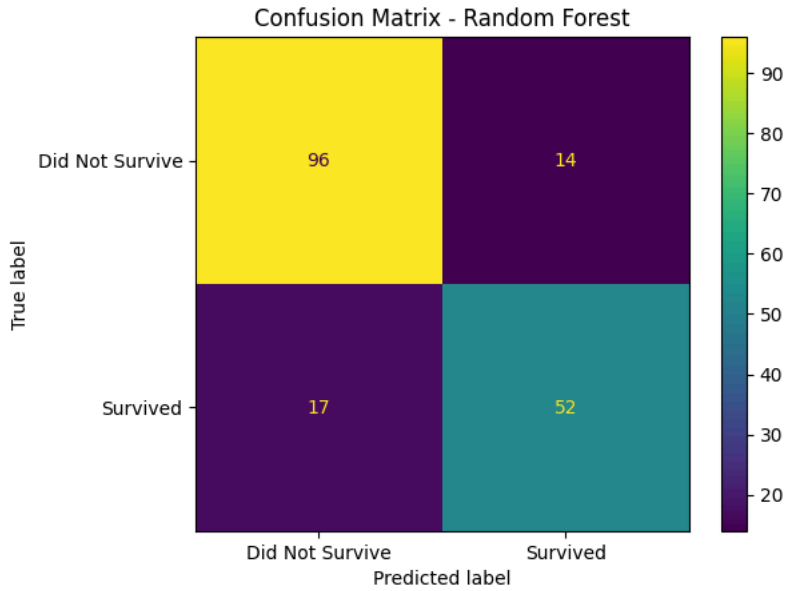
```
RandomForestClassifier
RandomForestClassifier(n_estimators=200, n_jobs=-1, random_state=42)
```

```
y_pred_log = log_reg.predict(X_test)
y_pred_rf  = rf.predict(X_test)
```

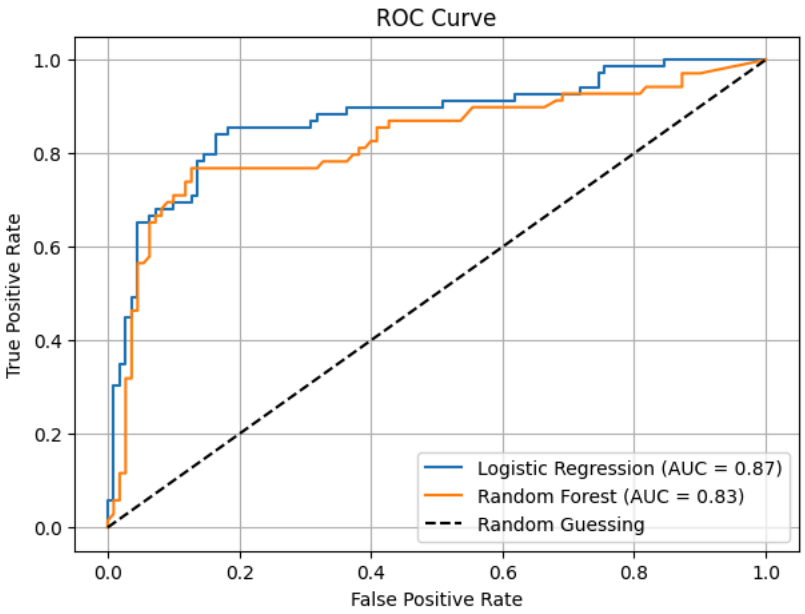
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

def print_metrics(y_true, y_pred, model_name):
    print(f"--- {model_name} ---")
    print("Accuracy :", accuracy_score(y_true, y_pred))
    print("Precision:", precision_score(y_true, y_pred))
    print("Recall   :", recall_score(y_true, y_pred))
    print("F1-Score :", f1_score(y_true, y_pred))
    print()

# Evaluate both models
print_metrics(y_test, y_pred_log, "Logistic Regression")
print_metrics(y_test, y_pred_rf, "Random Forest")
```

```
--- Logistic Regression ---
Accuracy : 0.8100558659217877
Precision: 0.7777777777777778
Recall   : 0.7101449275362319
F1-Score : 0.7424242424242424

--- Random Forest ---
Accuracy : 0.8268156424581006
Precision: 0.7878787878787878
Recall   : 0.7536231884057971
F1-Score : 0.7703703703703704
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Logistic Regression
cm_log = confusion_matrix(y_test, y_pred_log)
disp_log = ConfusionMatrixDisplay(confusion_matrix=cm_log, display_labels=["Did Not Survive", "Survived"])
disp_log.plot()
plt.title("Confusion Matrix - Logistic Regression")
plt.show()

# Random Forest
cm_rf = confusion_matrix(y_test, y_pred_rf)
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=["Did Not Survive", "Survived"])
disp_rf.plot()
plt.title("Confusion Matrix - Random Forest")
plt.show()
```

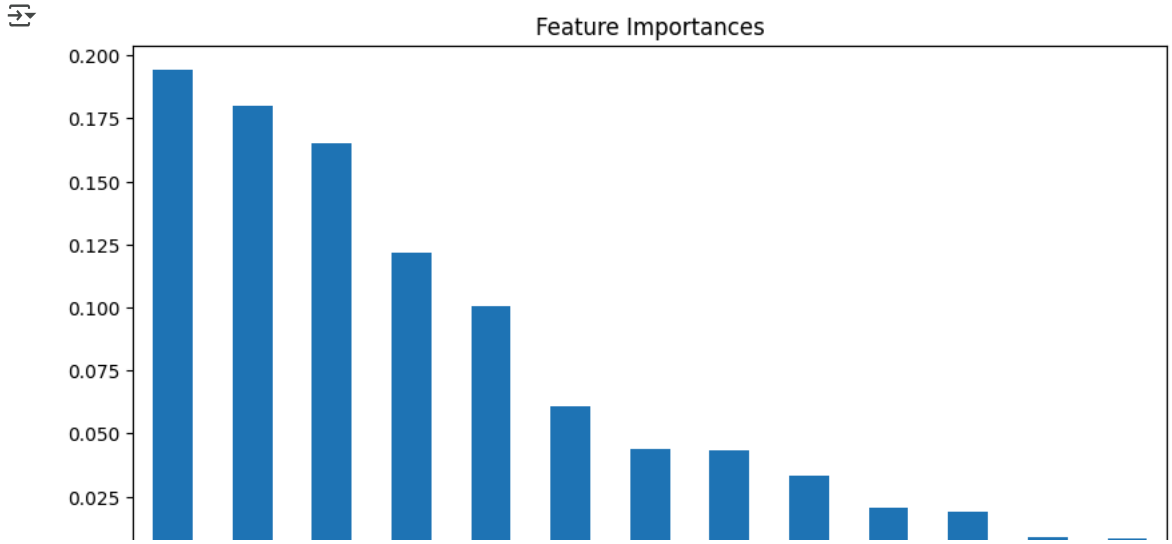

```
from sklearn.metrics import roc_curve, roc_auc_score

# Get predicted probabilities
y_prob_log = log_reg.predict_proba(X_test)[:,1]
y_prob_rf = rf.predict_proba(X_test)[:,1]

# Compute ROC Curve
fpr_log, tpr_log, _ = roc_curve(y_test, y_prob_log)
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_prob_rf)

# AUC scores
auc_log = roc_auc_score(y_test, y_prob_log)
auc_rf = roc_auc_score(y_test, y_prob_rf)

# Plot ROC Curve
plt.figure(figsize=(7,5))
plt.plot(fpr_log, tpr_log, label=f"Logistic Regression (AUC = {auc_log:.2f})")
plt.plot(fpr_rf, tpr_rf, label=f"Random Forest (AUC = {auc_rf:.2f})")
plt.plot([0,1], [0,1], 'k--', label='Random Guessing')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.grid(True)
plt.show()
```




```
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Initialize model
rf_model = RandomForestClassifier(random_state=42)

# Grid Search with 5-fold CV
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid,
                           cv=5, scoring='accuracy', n_jobs=-1, verbose=1)

grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.best_params_)
best_rf = grid_search.best_estimator_
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 200}
```

```
# Plot feature importances
import matplotlib.pyplot as plt
import seaborn as sns

feature_importances = pd.Series(best_rf.feature_importances_, index=X_train.columns)
feature_importances.sort_values(ascending=False).plot(kind='bar', figsize=(10,5), title="Feature Importances")
plt.show()

# Optionally remove least important features (e.g., below threshold)
selected_features = feature_importances[feature_importances > 0.01].index
X_train_sel = X_train[selected_features]
X_test_sel = X_test[selected_features]

# Retrain model on selected features
best_rf.fit(X_train_sel, y_train)
y_pred_rf_sel = best_rf.predict(X_test_sel)

from sklearn.metrics import accuracy_score
print("Accuracy after feature selection:", accuracy_score(y_test, y_pred_rf_sel))
```


Feature Importances
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025