Complete Code Guide: Customer Churn Prediction

Introduction and Overview

Code Explanation and Logic

Code Block 1:

```
# importing libraries and loading data set


import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
file_path = '/content/WA_Fn-UseC_-Telco-Customer-Churn.csv'
df = pd.read_csv(file_path)
```

Explanation:
This block performs the following operations:

Code Block 2:

```
df.head()
```

Explanation:
This block performs the following operations:

Code Block 3:

```
df.info()
df.describe()
```

Explanation:
This block performs the following operations:

Code Block 4:

```
statistics = df.describe(include='all')
missing_values = df.isnull().sum()

statistics, missing_values
```

Explanation:
This block performs the following operations:

Code Block 5:

```
data = {
    'customerID': ['7590-VHVEG', '5575-GNVDE', '3668-QPYBK', '7795-CFOCW', '9237-
HQITU'],
    'gender': ['Female', 'Male', 'Male', 'Male', 'Female'],
    'SeniorCitizen': [0, 0, 0, 0, 0],
    'Partner': ['Yes', 'No', 'No', 'No', 'No'],
    'Dependents': ['No', 'No', 'No', 'No', 'No'],
    'tenure': [1, 34, 2, 45, 2],
    'PhoneService': ['No', 'Yes', 'Yes', 'No', 'Yes'],
    'MultipleLines': ['No phone service', 'No', 'No', 'No phone service', 'No'],
    'InternetService': ['DSL', 'DSL', 'DSL', 'DSL', 'Fiber optic'],
    'OnlineSecurity': ['No', 'Yes', 'Yes', 'No', 'No'],
    'OnlineBackup': ['Yes', 'No', 'Yes', 'No', 'Yes'],
    'DeviceProtection': ['No', 'Yes', 'No', 'Yes', 'No'],
    'TechSupport': ['No', 'No', 'No', 'Yes', 'No'],
    'StreamingTV': ['No', 'No', 'No', 'Yes', 'Yes'],
    'StreamingMovies': ['No', 'No', 'No', 'No', 'No'],
```

```
    'Contract': ['Month-to-month', 'One year', 'Month-to-month', 'One year', 'Month-to-month'],
    'PaperlessBilling': ['Yes', 'No', 'Yes', 'No', 'Yes'],
    'PaymentMethod': ['Electronic check', 'Mailed check', 'Mailed check', 'Bank transfer (automatic)', 'Electronic check'],
    'MonthlyCharges': [29.85, 56.95, 53.85, 42.30, 70.70],
    'TotalCharges': ['29.85', '1889.5', '108.15', '1840.75', '151.65'],
    'Churn': ['No', 'No', 'Yes', 'No', 'Yes']
}
```

Explanation:
This block performs the following operations:

Code Block 6:

```
df_sample = pd.DataFrame(data)
df_sample['TotalCharges'] = pd.to_numeric(df_sample['TotalCharges'], errors='coerce')

df_sample_info = df_sample.info()
df_sample_statistics = df_sample.describe(include='all')
df_sample_missing_values = df_sample.isnull().sum()
df_sample_info, df_sample_statistics, df_sample_missing_values
```

Explanation:
This block performs the following operations:

Code Block 7:

```
df.hist(bins=30, figsize=(20, 15))
plt.show()

plt.figure(figsize=(20, 15))
sns.boxplot(data=df)
plt.show()
```

Explanation:
This block performs the following operations:

Code Block 8:

```
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
```

```
df['TotalCharges'].fillna(df['TotalCharges'].median(), inplace=True)
```

Explanation:
This block performs the following operations:

Code Block 9:

```
Q1 = df['MonthlyCharges'].quantile(0.25)
Q3 = df['MonthlyCharges'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df = df[(df['MonthlyCharges'] >= lower_bound) & (df['MonthlyCharges'] <= upper_bound)]
```

Explanation:
This block performs the following operations:

Code Block 10:

```
df['tenure_years'] = df['tenure'] / 12
df = pd.get_dummies(df, drop_first=True)
```

Explanation:
This block performs the following operations:

Code Block 11:

```
scaler = StandardScaler()
numerical_features = ['tenure', 'MonthlyCharges', 'TotalCharges']
df[numerical_features] = scaler.fit_transform(df[numerical_features])
```

Explanation:
This block performs the following operations:

Code Block 12:

```
X = df.drop('Churn_Yes', axis=1)
y = df['Churn_Yes']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Explanation:
This block performs the following operations:

Code Block 13:

```
#  random forest classifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
```

Explanation:
This block performs the following operations:

Code Block 14:

```
# Evaluate random forest classifier

metrics_rf = {
   "accuracy": accuracy_score(y_test, y_pred_rf),
   "precision": precision_score(y_test, y_pred_rf),
   "recall": recall_score(y_test, y_pred_rf),
   "f1_score": f1_score(y_test, y_pred_rf)
}
metrics_rf
```

Explanation:
This block performs the following operations:

Code Block 15:

```
# Logistic Regression
lr = LogisticRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
```

Explanation:
This block performs the following operations:

Code Block 16:

```
# Evaluate Logistic Regression model
metrics_lr = {
    "accuracy": accuracy_score(y_test, y_pred_lr),
    "precision": precision_score(y_test, y_pred_lr),
    "recall": recall_score(y_test, y_pred_lr),
    "f1_score": f1_score(y_test, y_pred_lr)
}

metrics_lr
```

Explanation of the Metrics:

1. **Accuracy**:
    - **Linear Regression Accuracy**: 0.8226 (~82.3%)
    - **Random Forest Accuracy**: 0.8048 (~80.5%)

Accuracy is the proportion of correct predictions out of the total number of predictions. In this case, Linear Regression has a slightly better accuracy (82.3%) compared to Random Forest (80.5%), meaning Linear Regression correctly predicted more instances overall.

2. **Precision**:
    - **Linear Regression Precision**: 0.6892 (~68.9%)
    - **Random Forest Precision**: 0.6929 (~69.3%)

Precision measures how many of the predicted positive cases were actually positive. Both models have similar precision (~69%), indicating they are equally good at avoiding false positives (incorrectly predicting something as positive when it's actually negative).

3. **Recall**:
    - **Linear Regression Recall**: 0.6005 (~60.1%)
    - **Random Forest Recall**: 0.4718 (~47.2%)

Recall measures how many of the actual positive cases were correctly predicted. Linear Regression has a significantly better recall (~60%) compared to Random Forest (~47.2%), meaning Linear Regression is better at identifying true positives, while Random Forest misses more actual positives.

4. **F1 Score**:
    - **Linear Regression F1 Score**: 0.6418 (~64.2%)
    - **Random Forest F1 Score**: 0.5614 (~56.1%)

The F1 score is the harmonic mean of precision and recall, providing a balance between the two. Linear Regression has a better F1 score (64.2%) compared to Random Forest (56.1%), indicating that Linear Regression strikes a better balance between precision and recall.

What This Means:

- **Linear Regression**:
    - This model has a higher **accuracy** (82.3%), meaning it makes more correct predictions overall.
    - The **precision** (68.9%) is slightly lower than Random Forest, but still comparable, meaning it is fairly reliable when it predicts something as positive.
    - The **recall** (60.1%) is better, indicating that it correctly identifies a higher number of actual positives compared to Random Forest.
    - Its **F1 score** (64.2%) reflects a better balance between precision and recall.
- **Random Forest**:
    - This model has slightly lower **accuracy** (80.5%) than Linear Regression.
    - It has a marginally higher **precision** (69.3%), meaning it is slightly better at avoiding false positives.
    - However, the **recall** (47.2%) is lower, indicating that Random Forest misses more true positives than Linear Regression.
    - The **F1 score** (56.1%) is lower, showing that it doesn't balance precision and recall as well as Linear Regression.

Summary:

- **Linear Regression** seems to perform better overall, particularly in terms of recall and F1 score, indicating it is better at identifying true positives and balancing precision and recall.

- **Random Forest** has a slightly better precision, which means it's more careful in avoiding false positives, but it sacrifices recall, missing more true positives.