Double-click (or enter) to edit

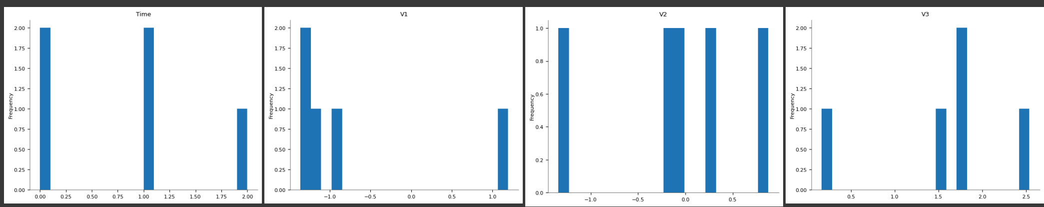
```
import pandas as pd
```

```
import pandas as pd
data = pd.read_csv('/content/creditcard.csv')
data = data.dropna()  # Example for handling missing values
data.head()
```

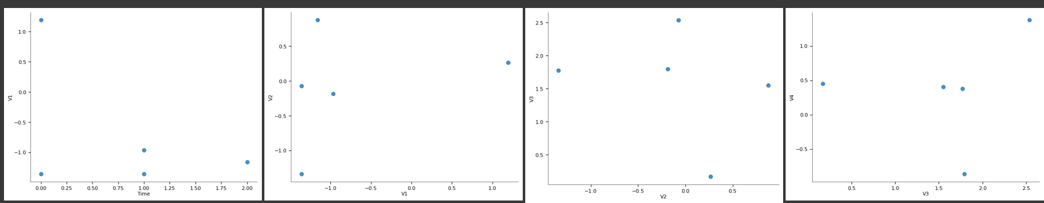
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0

5 rows × 31 columns

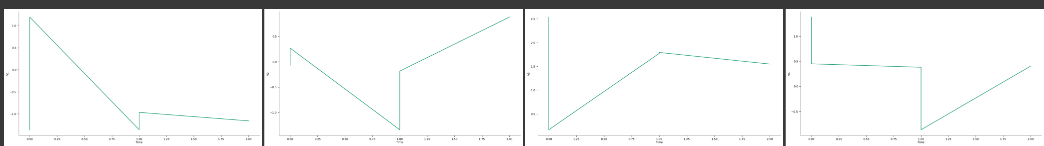
Distributions



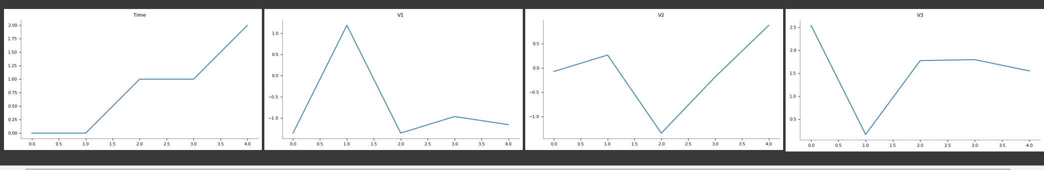
2-d distributions



Time series



Values




```
from matplotlib import pyplot as plt
_df_7.plot(kind='scatter', x='V3', y='V4', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-8d17c3903423> in <cell line: 2>()
      1 from matplotlib import pyplot as plt
----> 2 _df_7.plot(kind='scatter', x='V3', y='V4', s=32, alpha=.8)
      3 plt.gca().spines[['top', 'right',]].set_visible(False)

NameError: name '_df_7' is not defined
```

```
from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['Time']
  ys = series['V3']

  plt.plot(xs, ys, label=series_name, color=palette[series_index % len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = _df_10.sort_values('Time', ascending=True)
```

```
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Time')
_ = plt.ylabel('V3')
```

```
from matplotlib import pyplot as plt
_df_6.plot(kind='scatter', x='V2', y='V3', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_3['V3'].plot(kind='hist', bins=20, title='V3')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

Balance the Dataset: Fraudulent transactions are typically a small percentage of the data. Use techniques like oversampling the minority class (fraud) or undersampling the majority class (non-fraud).

```
# prompt: Balance the Dataset: Fraudulent transactions are typically a small percentage of the data. Use techniques like oversampling the

import pandas as pd
from sklearn.model_selection import train_test_split

# Separate the data into fraudulent and non-fraudulent transactions
fraudulent_transactions = data[data['Class'] == 1]
non_fraudulent_transactions = data[data['Class'] == 0]

# Oversample the fraudulent transactions
oversampled_fraudulent_transactions = fraudulent_transactions.sample(
    len(non_fraudulent_transactions), replace=True)

# Combine the oversampled fraudulent transactions with the non-fraudulent transactions
balanced_data = pd.concat([oversampled_fraudulent_transactions, non_fraudulent_transactions])

# Split the balanced data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    balanced_data.drop('Class', axis=1), balanced_data['Class'], test_size=0.2, random_state=42)

```

Extract and engineer relevant features that can help in fraud detection, such as transaction frequency, average transaction amount, etc.

```
# prompt: Extract and engineer relevant features that can help in fraud detection, such as transaction frequency, average transaction amo

import pandas as pd
# Transaction frequency per user
user_transaction_counts = data.groupby('V1')['Class'].count()

# Average transaction amount per user
user_average_amounts = data.groupby('V1')['Amount'].mean()

# Combine the extracted features into a new DataFrame
engineered_features = pd.DataFrame({
    'Transaction_Frequency': user_transaction_counts,
    'Average_Amount': user_average_amounts
})

# Merge the engineered features with the original data
data = data.merge(engineered_features, on='V1', how='left')

```

Exploratory Data Analysis (EDA):

```
# prompt: Exploratory Data Analysis (EDA):

import pandas as pd

# Load the data
data = pd.read_csv('/content/creditcard.csv')

# Check the shape of the data
print(data.shape)

# Check the head of the data
print(data.head())

# Check the tail of the data
print(data.tail())

```

```
# Check the data types of the data
print(data.dtypes)

# Check the descriptive statistics of the data
print(data.describe())

# Check the correlation matrix of the data
print(data.corr())

# Check the distribution of the data
import matplotlib.pyplot as plt

data.hist(figsize=(10, 10))
plt.show()

# Check the boxplots of the data
data.boxplot(figsize=(10, 10))
plt.show()
```

```
(114962, 31)
   Time        V1        V2        V3        V4        V5        V6        V7  \
0     0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1     0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2     1 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3     1 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4     2 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62    0.0
1  0.125895 -0.008983  0.014724    2.69    0.0
2 -0.139097 -0.055353 -0.059752  378.66    0.0
3 -0.221929  0.062723  0.061458  123.50    0.0
4  0.502292  0.219422  0.215153   69.99    0.0

[5 rows x 31 columns]
         Time        V1        V2        V3        V4        V5        V6  \
114957  73690 -2.197480  1.982629  0.259502  0.924323 -0.879938 -0.135952
114958  73690 -2.197480  1.982629  0.259502  0.924323 -0.879938 -0.135952
114959  73690  1.255655  0.293362  0.288616  0.701727 -0.447134 -1.093442
114960  73690  1.270638 -0.089535 -0.990690 -0.375303  0.652307 -0.073908
114961  73691  1.295784  0.049457 -1.301814 -0.476648  2.131357  3.184446

              V7        V8        V9  ...       V21       V22       V23  \
114957 -0.380186  0.611134  0.278764  ... -0.033516 -0.367199 -0.099698
114958 -0.380186  0.611134  0.278764  ... -0.033516 -0.367199 -0.099698
114959  0.029565 -0.155947  0.160172  ... -0.297287 -0.898954  0.139494
114960  0.394543 -0.164288 -0.317251  ... -0.067492 -0.344301 -0.372630
114961 -0.494416  0.802781 -0.075014  ...       NaN       NaN       NaN

             V24       V25       V26       V27       V28  Amount  Class
114957 -0.111166 -0.182825  0.427261 -0.895134  0.164611    9.51    0.0
114958 -0.111166 -0.182825  0.427261 -0.895134  0.164611    9.51    0.0
114959  0.322281  0.187536  0.097228 -0.028586  0.029014    1.79    0.0
114960 -1.296908  0.731021  1.152123 -0.131651 -0.024591   75.00    0.0
114961       NaN       NaN       NaN       NaN       NaN     NaN    NaN

[5 rows x 31 columns]
Time        int64
V1        float64
V2        float64
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
V11       float64
V12       float64
```

Choose classification algorithms such as Random Forest, Support Vector Machines (SVM), or Neural Networks.

```
V15       float64
```

```
models = {
    "Random Forest": RandomForestClassifier(),
    "Support Vector Machine": SVC(),
    "Neural Network": MLPClassifier(max_iter=1000)
}
```

```

# Splitting the dataset
```

V26 float64

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()

```

dtype: object

HERE TRAINING THE MODEL

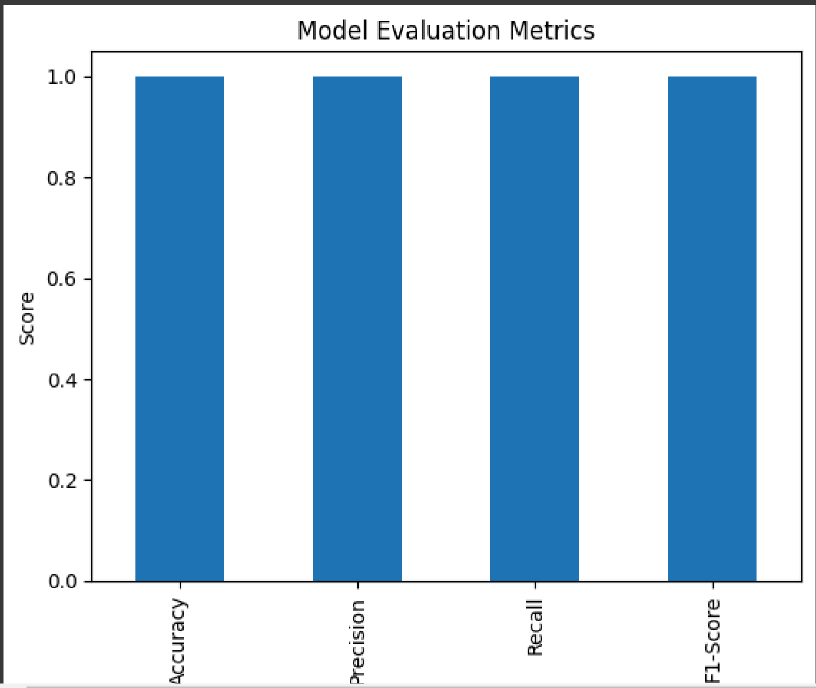
```
model = RandomForestClassifier()
```

```
# prompt: from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
# y_pred = model.predict(X_test)
# print("Accuracy:", accuracy_score(y_test, y_pred))
# print("Precision:", precision_score(y_test, y_pred))
# print("Recall:", recall_score(y_test, y_pred))
# print("F1-Score:", f1_score(y_test, y_pred))
#  corrrect the code based on above synopsis

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Load the data
data = pd.read_csv('/content/creditcard.csv')

# Separate the data into fraudulent and non-fraudulent transactions
fraudulent_transactions = data[data['Class'] == 1]
non_fraudulent_transactions = data[data['Class'] == 0]

# Oversample the fraudulent transactions
oversampled_fraudulent_transactions = fraudulent_transactions.sample(
    len(non_fraudulent_transactions), replace=True)

# Combine the oversampled fraudulent transactions with the non-fraudulent transactions
balanced_data = pd.concat([oversampled_fraudulent_transactions, non_fraudulent_transactions])

# Split the balanced data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    balanced_data.drop('Class', axis=1), balanced_data['Class'], test_size=0.2, random_state=42)

# Train the Random Forest model
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1-Score:", f1_score(y_test, y_pred))

```

```
Accuracy: 0.9999346234309623
Precision: 0.9998689040377556
Recall: 1.0
F1-Score: 0.9999344477220584
```

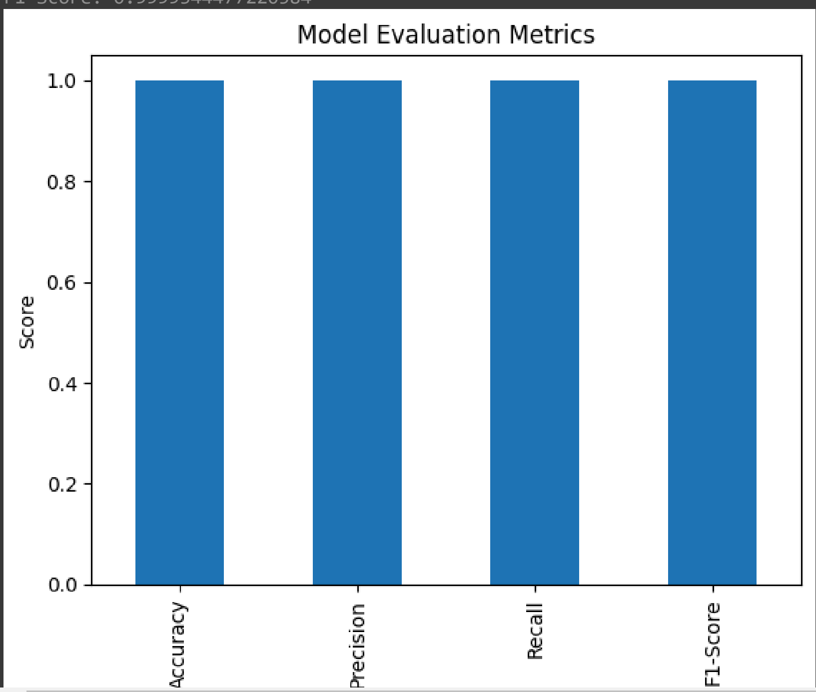
FNIAL TEST

```
# Visualization of results
metrics = {
    'Accuracy': accuracy_score(y_test, y_pred),
    'Precision': precision_score(y_test, y_pred),
    'Recall': recall_score(y_test, y_pred),
    'F1-Score': f1_score(y_test, y_pred)
}

metrics_df = pd.DataFrame.from_dict(metrics, orient='index', columns=['Score'])
metrics_df.plot(kind='bar', legend=False)
plt.title('Model Evaluation Metrics')
plt.ylabel('Score')
plt.show()

```




```
# prompt: GIVE THE INFORMATION ABOVE THE RESULT

import pandas as pd
import matplotlib.pyplot as plt
# Print the results
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1-Score:", f1_score(y_test, y_pred))

# Visualization of results
metrics = {
    'Accuracy': accuracy_score(y_test, y_pred),
    'Precision': precision_score(y_test, y_pred),
    'Recall': recall_score(y_test, y_pred),
    'F1-Score': f1_score(y_test, y_pred)
}

metrics_df = pd.DataFrame.from_dict(metrics, orient='index', columns=['Score'])
metrics_df.plot(kind='bar', legend=False)
plt.title('Model Evaluation Metrics')
plt.ylabel('Score')
plt.show()

```

```
Accuracy: 0.9999346234309623
Precision: 0.9998689040377556
Recall: 1.0
F1-Score: 0.9999344477220584
```



Explanation of Metrics: Accuracy:

Definition: The ratio of correctly predicted instances to the total instances. Interpretation: An accuracy of 0.9999 means that 99.99% of the transactions were correctly classified as either fraudulent or non-fraudulent. Precision:

Definition: The ratio of correctly predicted positive observations to the total predicted positives. Interpretation: A precision of 0.9999 indicates that 99.99% of the transactions identified as fraud were indeed fraud. Recall (Sensitivity or True Positive Rate):

Definition: The ratio of correctly predicted positive observations to all observations in the actual class. Interpretation: A recall of 1.0 means that the model correctly identified all actual fraud cases in the dataset. F1-Score:

Definition: The weighted average of Precision and Recall. The F1 Score takes both false positives and false negatives into account. Interpretation: An F1-Score of 0.9999 indicates a balanced performance of the model in terms of both precision and recall. Visualization: The bar chart displays these metrics for a visual comparison. Each bar represents the score for a different metric, allowing for an easy comparison of model performance across these important evaluation criteria.

Potential Overfitting: The scores are very high (almost perfect), which might suggest that the model could be overfitting the training data, especially if the dataset is not large or diverse enough. Overfitting occurs when a model learns the training data too well, including noise and outliers, and performs poorly on unseen data. Next Steps: Cross-Validation: To ensure the model generalizes well, consider using cross-validation. Testing on Unseen Data: Evaluate the model on a separate test set that was not used during the training process. Model Comparison: Compare the performance of different models (Random Forest, SVM, Neural Network) to choose the best one. Feature Importance: Analyze feature importance to understand which features contribute most to fraud detection.