Dataanalyzer.py

```
from abc import ABC, abstractmethod

class AnalysisError(Exception):
	pass

class DataAnalyzer(ABC):
	@abstractmethod
	def analyze(self, data):
	pass

class TextDataAnalyzer(DataAnalyzer):
	def analyze(self, data):
	try:
	if not isinstance(data, str):
		raise TypeError("Data must be of type str for text analysis.")

	word_count = len(data)
	print(f"Text analysis complete. Word count: {word_count}")

	except TypeError as e:
	print(f"TypeError encountered: {e}")

	except AnalysisError as e:
	print(f"AnalysisError encountered: {e}")


class NumericDataAnalyzer(DataAnalyzer):
	def __init__(self, funding_goal):
	self.funding_goal = funding_goal

	def analyze(self, data):
	try:
	if not isinstance(data, (int, float)):
		raise TypeError("Data must be a number for numeric analysis.")

	# Check funding goal
	if data >= self.funding_goal:
		print(f"Funding goal of {self.funding_goal} achieved!")
	else:
		print(f"Funding goal not met. Current funding: {data}")

	except TypeError as e:
	print(f"TypeError encountered: {e}")
	except ValueError as e:
	print(f"ValueError encountered: {e}")
```

```
        except KeyError as e:
        print(f"KeyError encountered: {e}")
        except AnalysisError as e:
        print(f"AnalysisError encountered: {e}")

# Test the implementation
def main():
        text_analyzer = TextDataAnalyzer()
        numeric_analyzer = NumericDataAnalyzer(funding_goal=1000)

        analyzers = [text_analyzer, numeric_analyzer]

        data_entries = ["This is a sample text for analysis.", 1500, "Another text entry.", 500,
1000]

        for analyzer in analyzers:
        for data in data_entries:
        print(f"\nAnalyzing with {analyzer.__class__.__name__}:")
        analyzer.analyze(data)

if __name__ == "__main__":
        main()
```

creatorsupport.py

```
from abc import ABC, abstractmethod

class Supporter(ABC):
        @abstractmethod
        def support(self):
        pass

class Donor(Supporter):
        def __init__(self,name,amount):
        self.name = name
        self.amount = amount
        def support(self):
        print(f"{self.name} has donated {self.amount} $ towards the project")

class Subscriber(Supporter):
        def __init__(self,name,months):
        self.name = name
        self.months = months
        def support(self):
        print(f"{self.name} has subscribed for {self.months} months")
```

```
class Funder(Supporter):
        def __init__(self,name,funds):
        self.name = name
        self.funds = funds
        def support(self):
        print(f"{self.name} has funded the project with {self.funds} $")


_donor = Donor(name = "Sam",amount=40)
_subscriber = Subscriber(name = "Jeff",months=5)
_funder = Funder(name = "Charlie", funds=5000)

supporter_all = [_donor, _subscriber, _funder]

for _supporter in supporter_all:
        _supporter.support()
```